University of St Andrews

School of Computer Science

Appendix VI

Samples of Final Year Projects with Marking Sheets

Automated Class Questionnaires – Acquire

Author: Gareth Edwards

University of St Andrews

24th April 2003

Abstract

This document discusses the Acquire system designed to facilitate on-line submissions of module reviews by students enrolled at the University of St Andrews; specifically the members of the School of Computer Science. The current system requires students to fill in and submit paper based forms which are read by an optical system. This system has proved unreliable so the purpose of this project was to create a new computerised system prototype to investigate whether replacing the existing system with a web-based form submission system was viable and to discover any advantages and disadvantages of such a system.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is 14004 words long, including project specification and plan.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain copyright in this work.

Gareth Edwards

1 INTRODUCTION 6

1.1 PROJECT GOAL 6
1.2 THE EXISTING SYSTEM 6
1.3 ACQUIRE 7

2 PROJECT DETAILS 8

2.1 CHANGES TO THE PROJECT PLAN 8
2.2 OVERVIEW OF SYSTEM STRUCTURE 8
2.3 AREAS OF PARTICULAR INTEREST 9
2.3.1 Ensuring anonymity 9
2.3.1.1 How the system works 10
2.3.2 Form display 11

3 EVALUATION AND CRITICAL APPRAISAL 12

3.1 EVALUATION AGAINST ORIGINAL OBJECTIVES 12
3.1.1 Data collection 13
3.1.2 Output 14
3.1.3 Anonymity 15
3.1.4 Customisable 15
3.1.5 User Interface 17
3.1.6 Efficient 18
3.1.7 Security 20
3.1.8 Maintainable 22
3.1.9 Scalable 22
3.2 EVALUATION AGAINST RELATED WORK BY OTHERS 23
3.3 EVALUATION AGAINST SIMILAR WORK IN THE PUBLIC DOMAIN 23

4 CONCLUSIONS 24

5 APPENDICES 26

5.1 PROJECT OBJECTIVES 26
5.2 REQUIREMENTS SPECIFICATION – VERSION 1.0 27
5.2.1 Preface 27
5.2.1.1 Product Name: Acquire 27
5.2.1.2 Version History 27
5.2.1.3 Intended Audiences 27
5.2.2 User Requirements Definition 28
5.2.2.1 Web-based questionnaires 28
5.2.2.2 Authentication 28
5.2.2.3 Anonymity 29
5.2.2.4 Questionnaire Contents 29

5.2.2.5 Output 30
5.2.2.6 Help Details 30
5.2.3 System Architecture *31*
5.2.3.1 Web Interfaces 31
5.2.3.2 Database 32
5.2.3.3 SQL 32
5.2.4 System Requirements Definition *33*
5.2.4.1 System Implementation 33
5.2.4.2 Reuse 33
5.3 DESIGN AND IMPLEMENTATION 33
5.3.1 Development Methods *33*
5.3.1.1 Process Model 33
5.3.1.2 Implementation Tools & Languages 34
5.3.2 Project Management *35*
5.3.2.1 Change Management 35
5.3.2.2 Version Control 36
5.3.2.3 Deadlines and Deliverables 36
5.3.2.4 Milestones 37
5.3.3 Resources *37*
5.3.3.1 Hardware 37
5.3.3.2 Software 38
5.3.3.3 Resource Constraints 38
5.3.4 Risks and Fall-back Plans *39*
5.3.5 Quality Control *39*
5.4 TESTING 40
5.4.1 Test Plan *40*
5.4.1.1 Black-box testing 40
5.4.1.2 Modular testing 40
5.4.1.3 Testing the database 41
5.4.1.4 Stress testing 41
5.4.1.5 Security Testing 41
5.5 PROJECT MONITORING SHEET 43
5.6 INTERIM REPORT 1 44
5.6.1 Design Decisions *44*
5.6.2 Schedule *44*
5.6.3 Other Notes *44*
5.7 INTERIM REPORT 2 45
5.7.1 User Authentication *45*
5.7.2 Design Changes *45*
Database: 45
Design environment: 45
5.7.3 Schedule *45*
5.8 LIST OF CHANGES 46

5.9 TESTING SUMMARY 47
5.9.1 Testing Form Display *47*
5.9.2 Testing the Database *48*
5.9.3 Testing the Tomcat Server *48*
5.9.4 Testing the security *49*
5.10 STATUS REPORT 49
5.10.1 Major Contributions *50*
5.10.2 Deficiencies *50*
6 GLOSSARY 51
7 REFERENCES 51
8 ADDITIONAL APPENDICES 51

[Due to the large number of acronyms present in this document, all acronyms are listed in a separate glossary section of the appendices rather than explained in-line.]

1 Introduction

1.1 Project goal

The goal of this project was to create a web-based system which allows students enrolled within the university to submit reviews of studied modules accurately and easily, and to allow relevant university staff (module coordinators, lecturers etc) to view and analyse the results of the reviews. In accordance with university policy and the Data Protection Act, the reviews should be anonymous: that is, it should be impractical for anyone to be able to link a particular review to a particular student, and also that the system must not insecurely store any confidential or sensitive information about its users.

The system was not intended to be production class system but rather a prototype to prove whether such a system could be created; to discover any advantages and disadvantages of such a system; to discover any problems likely to be encountered if such a production class system was to be developed and to detail potential solutions to the problems.

1.2 The existing system

At the end of each semester, students are asked to complete one or more forms (depending upon whether they are an honours or sub-honours student) reviewing the module or modules they have studied in a number of criteria. The forms typically ask questions regarding the standard of lectures, how helpful tutors were, quality of handouts etc, and also questions about how the student thinks he/she has performed during the module. Finally students are able to submit general comments about the module.

The existing forms use optical mark reading technology. The student is asked to place a black mark in a box corresponding to the intended answer and the forms are then passed through an optical reader which collects the results. This system is flawed in a number of areas.

Firstly, the system is inherently error prone in that optical reading offers between 95-98% accuracy for reads, which considering each of the 5508 undergraduates [1] completes on average 3 forms with 12 questions, means that between 4,000 to 10,000 questions will be incorrectly recorded each semester. This figure doesn't take into account forms which are incorrectly filled in, or which are completed in such a way that the system cannot read a result at all, be it correct or otherwise.

Secondly, the forms themselves are frequently ambiguous, leading to students submitting an unintended answer. This ambiguity manifests itself in a number of ways, but the main cause is inconsistency between answer fields. One question may indicate the use a higher number to represent a positive response, but for the next question a lower number is better. Unless the student reads the form very closely it is easy to submit the exact opposite of the intended answer. The existing system has no way of knowing that the student meant to submit a different answer so even the results which are correctly read by the system cannot necessarily be assumed to be correct.

Finally, there is no way to store general comments submitted on the forms, other than by storing the paper forms physically. While this isn't a problem, it is a significant limitation of the existing system.

1.3 Acquire

The prototype system developed during the course of this project has been named Acquire, for no reason other than it was the only relevant word based around the three initials of "Automated Class Questionnaires" – the official project title.

The project successfully met eight of its nine objectives. The remaining objective which was not met 100% successfully was that of form customisability. Why this objective was not met is discussed in full in the "Evaluation and Critical Appraisal" section of this report.

The most important objectives of the prototype were that the forms should be easy to understand and complete; the forms should be unambiguous; the results must be

reliable and student anonymity must be maintained at all times. Each of these requirements was met successfully and is discussed in greater detail in the next section of this document.

2 Project Details

2.1 Changes to the project plan

In the original project plan it was stated that Oracle would be used to power the back-end database unless licensing issues regarding its use could not be rectified. However, although there were no licensing problems with Oracle, a decision was made with consent from the customer to change the database from Oracle to MySQL in order to reduce complexity within the system. Oracle is an extremely powerful product and has correspondingly large administration overheads.

A typical Oracle database will be maintained by number of people, each with a very specific and difficult job requiring lengthy training. It was simply impossible for the single developer of Acquire to learn and perform each of these jobs in the available time, nor would it be satisfactory to the customer to have to do the same. MySQL is also a very powerful database system but significantly easier to use for most applications. Although it does lack some functionality available in products such as Oracle, none of these functions were required by the Acquire system (one desired property of the system was that it would not be dependent on any one single product) and as MySQL is freely available for non-commercial use and has a long history of operating as web-site back-end databases it was the natural choice to replace Oracle.

2.2 Overview of system structure

The Acquire system can be broken down into five main process areas: authentication, form generation, form display, submission processing, and results analysis.

Which areas are available to a particular user depends upon their role within the system, be it student or staff.

Students log into the system and are presented with a page allowing them to submit new reviews, or update previous reviews, which is the authentication section. Once

the student has chosen a module to review they are taken to the appropriate form which makes up the form display area. Upon submitting the form it is passed to the form processing area which extracts the relevant information and updates the database accordingly. These are the only sections available to the students.

Staff log into the system and are presented with a page listing the modules that they can create question forms for, and also links to the result analysis section. If a staff member has already created forms for each module then they will only be presented the links to the results analysis section because forms cannot be edited once created. This limitation is necessary because if a staff member changed the review form after students had started submitting reviews then the database would become inconsistent and results could not be trusted.

The form creation and results analysis sections are different depending upon whether currently selected module is an honours or sub-honours module and only allow choices relevant to the particular form type to be made.

2.3 Areas of particular interest

2.3.1 Ensuring anonymity

One of the most important requirements and objectives of the system was that it should be impractical for anyone other than the system administrator to be able to link a particular set of form answers to the student that submitted them. This objective was not only met, but actually surpassed because it is not only impractical for module coordinators to link answers to students, it is essentially impossible. In addition, it has been made impractical for even the system administrator to link the two.

Because a requirement of the system was that it must be possible for students to update their answers as they see fit it was necessary to store details about which answers were submitted by which user. Clearly this meant that anyone with access to the database would be able to extract details about which answers were sent by a student, which meant that the system was not anonymous.

The initial solution was that special constraints would be placed on users via the authentication mechanisms provided by MySQL which would restrict the type of queries runnable by users depending upon their credentials. However, the root user, or system administrator would still be able to run any type of query, and ultimately strong links would still exist between users and their answers within the database.

The delivered system solved each of these problems by taking an entirely new approach. Instead of making it difficult to access the links between students and answers, the links are easily visible to anyone that's interested but the details about the student are stored in such a way that makes the links useless to anyone other than the system itself. This is a achieved by storing the user details as Message Authentication Codes (MAC) generated using the SHA-1 secure hash algorithm [2] in conjunction with a secret key. This key is stored in a secure key store accessible only via a password known only to the system and the system administrator.

2.3.1.1 How the system works

The user logs in using their regular username and password. Once they have been verified as a user of the system their unique hash code is generated using the SHA-1 algorithm and the secret key operating on the username. The value generated is a string but to avoid compatibility problems caused by the presence of " " characters in SQL code the value is converted to a hexadecimal integer, a process which doesn't affect its security in any way. From this point onwards, the system only ever uses this hash value to identify the student and when the student submits a form this hash value is stored in the database alongside the answers.

The next time the student logs in, the hash value is regenerated and in this way the system is able to retrieve the student's answers without ever compromising anonymity.

By employing this system, it becomes impossible for anyone other than the system administrator to links students to answers as they do not have access to the required secret key. Because the system administrator has total control over the system and knows the key store password it is possible that they could generate the correct hash value. However, to do so they would first have to write a program which can extract

the secret key from the key store, generate the hash value using the SHA-1 algorithm and then run queries on the database. This isn't particularly difficult but it isn't particularly useful either. This method would allow the system administrator to see the answers submitted by a particular student, but not to see *which* answers were submitted by *which* student – a subtle but important difference because it is possible that a staff member would like to know who submitted a bad review, but they probably wouldn't be interested in just seeing the answers given by a particular person.

The technical details of the SHA-1 algorithm are very complex and beyond the scope of this document, but it is an algorithm designed by NIST and the NSA and which has survived a decade of cryptanalysis so its security can be relied upon for this system. The NIST web page giving full details of the algorithm can be found at reference [2] at the end of this document.

2.3.2 Form display

Question forms are generated dynamically at run-time via a combination of technologies. The basic structure of the forms and the questions they contain is determined when the form is created by the appropriate staff member, and this form is stored as an XML file which doesn't contain any formatting information. For the form to be displayed in a usable way in all web browsers it needs to be converted from XML to HTML, and the correct formatting applied. The initial transformation from XML to HTML is performed using XSLT, a specialist language used for transforming XML data. The XSLT is applied to the XML file when the form is requested, and the transformation is performed by the Xalan XSLT processor. The final formatting is performed in the user's browser using CSS, which requires that the user's browser supports CSS. The requirements specification stated that the system should generate pages according to the latest industry standards, and the latest W3C standard, HTML4.01 (XHTML 1.0) requires CSS support so it can be assumed. However, if the browser doesn't support CSS then the whole system will still work, but with some formatting missing, none of which will affect the ability to submit forms or display the results.

This system greatly simplifies the code required to display the forms and means that many aspects of the system appearance can be altered easily without the need to recompile any program code as the XSLT source files are simple XML files editable in any text-editor by anyone that understands XSLT. However, while the use of XML and XSLT increases flexibility in some respects, it also has some serious disadvantages relating to the customisability of the basic form structure. These problems are the basis of the one objective not successfully met and are discussed in depth in the following section "Evaluation and Critical Appraisal".

3 Evaluation and Critical Appraisal

3.1 Evaluation against original objectives

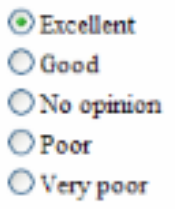
At the outset of the project nine primary objectives for the project were identified, and if fulfilled then the project would be deemed a success. The nine objectives, in approximate order of importance, were as follows:

- Data collection – Accurate results must be stored in a usable way.
- Output – Generate output more reliable than the existing system.
- Anonymity – It must be impractical for an interested party to link a particular set of answers to a particular student.
- Customisable – It must be easy to change the questions asked on forms.
- User-interface – Only the most basic computer literacy must be assumed of all users.
- Efficient – The system must be efficient in terms of general access and database operations.
- Security – The system must not allow users to learn other user's password or confidential details. Password must be encrypted before being sent across the Internet.
- Maintainable – It should be possible for a future programmer to fix bugs, and add or change functionality without affecting unrelated parts of the system.
- Scalable – The system should grow as student numbers and/or module numbers grow.

What follows is an evaluation of these objectives against what was actually achieved by the project.

3.1.1 Data collection

In order for this objective to be met it was necessary to accurately collect and store all data submitted by the students when reviewing modules. It was essential that the forms presented to the students were easy to understand and totally free of ambiguity. To ensure that all ambiguity was removed, the current system whereby a student chooses an integer corresponding to their opinion was discarded as it was inherently ambiguous and the source of many of the errors found the in existing system. Instead, all possible answers were listed using standard, easily understood terms to express an opinion. For example, if the following question was asked, "How do you rate this module overall?" the student would be able to choose from the following answers:



The form processor then converts this into a suitable integer for storage in the database, but does so in a consistent way which means that the results can be trusted as accurate. The system knows which integer maps to which answer so the results analysis can display suitable details regarding the question asked and the answers given.

Although not all questions required "Excellent | Good | No Opinion | Poor | Very Poor" type answers, each question was coupled with a suitable answer type and the integer map maintained so every question could be answered in a totally unambiguous way and the results displayed appropriately.

The second part of this objective was that the results must be stored in a usable way. The results are stored in a codified way in the database in such a way that the results analysis section can interrogate the database and get accurate results relating to the module and questions requested. Any system which understood the coding in use would be able to interrogate the database and produce its own results. The database

design is not perfect and if another section of the system (form generation) had been implemented differently (detailed in the Customisability objective review later) then the database solution would have been more elegant, but nonetheless it does the required job of storing the data in an accurate and usable way.

As both of the criteria governing the data collection objective have been met it is fair to say that this objective was fulfilled. Certain aspects, especially relating to the database would be done differently if another version of the system was to be created but it still does everything it is supposed to do.

3.1.2 Output

Originally this objective involved two separate criteria: that the system should produce output more reliable than the existing system, and that the raw data should be available to module coordinators so that they could confirm the results are correct. During the course of the project it was decided with the customer that it was not necessary to provide the raw data as long as development testing showed the results to be accurate, because the codified nature of the database meant that humans could not interpret the data manually, or at least it would be impractical to do so.

The output produced by the existing system is frequently incorrect, and even when correct it is often not clear what a set of results actually mean. The results make use of an unusual, and inaccurate graph-like structure which gives an approximation of the results, as well as an average result (an example of the current output can be found in the appendices). This system is not at all satisfactory and was in no way replicated by the Acquire system.

Acquire simply lists the question, the possible answers to a question, and the percentage returned for each particular answer, as well as the actual number of users giving that answer. For example, for a module with 200 reviews:

How do you rate this module overall?

Excellent	Good	No opinion	Poor	Very poor
25% (50)	35% (70)	10% (20)	25% (50)	5% (10)

This is far simpler than the existing system, but still manages to convey much more useful information. Armed with this information it is much easier to quickly gauge the general feeling towards the question topic than with the existing output.

The system adequately meets the criterion specified as part of this objective and so the objective is fulfilled.

3.1.3 Anonymity

To fulfil this objective it must be impractical for any module coordinator or other interested party to learn the origin of a particular submission or other confidential details regarding users.

This objective has already been discussed in section 2.3.1 “Ensuring anonymity” above. As explained in that section, the system guarantees anonymity of all its users by representing all users by a 20-byte unique secure hash value based on a SHA-1 MAC generated using a secret key.

This objective has not only been met but in fact surpassed so it can definitely be considered fulfilled.

3.1.4 Customisable

This objective required that it must be easy to change the data regarding module details and questions.

The system offers a degree of customisability of the sub-honours forms in that it is possible to choose which questions will appear on a question form, but the questions must come from a predefined list. Honours forms have very little customisability in that the only things which can be customised is the list of honours modules that will be listed on the form; other than that the questions are standard. To change the questions that are asked on the forms would require significant editing of Servlet source code and recompilation as the structure of the files is hard-coded into the system. This is unsatisfactory but is the result of a design decision taken early on in the project design which didn’t reveal itself to be flawed until too late in the

development to be changed as its impact would have required an almost total rewrite of the system.

As discussed earlier in the document, the system stores the question form for a particular module as an XML file, which is dynamically converted to HTML using the XSLT language. This is an elegant way of solving the formatting problem – that is, it totally separates the content from presentation so the appearance of the forms can be changed easily without having to change the structure of the data. However, the XSLT formatting code has to be written to a particular structure known in advance, so the structure of the XML files must be consistent and predictable. Secondly, because question identifiers on each form must be known if they are to used in the database to run queries so these must be predetermined. A question is identified in the data by three fields, module code e.g. CS2001, general question type e.g. summary, and the specific question index e.g. 3 (the third question of the summary section). Using this simple system in conjunction with the known structure of the XML file means that it is easy to link results with questions and to format them accordingly.

By relying on hard-coded knowledge of the files, it becomes very difficult to change the available questions and the way the forms are processed, which essentially eliminates customisability. This constitutes a large flaw in the system.

If the system was implemented again, with the benefit of experience, then the system would become 100% database driven, with all data regarding question forms and question identifiers stored as fields in the database. This would remove the hard-coded dependencies of the XML files because the module coordinator that creates the form could specify their own questions and a dynamic identifier could be created which linked the question asked to the answers submitted. Making such a change would have a large impact on the design of the system and would make the formatting of the pages much more difficult, but would allow the ease of customisability of forms.

This flawed design came about because of the order in which the system was built. The first tasks in the schedule were related to the design and construction of the question form structure and how to format them for display. What appeared to be the

best solution to these problems was chosen and implemented without due consideration of how it would affect development of later sections. When it became time to implement the form generators for use by staff and to link the contents in the database it became apparent that the model chosen was highly restrictive but it was too late to be rectified without jeopardising the project as a whole.

This objective cannot be considered fulfilled. Although there is an element of customisability available, it is nowhere near the extent required for successful fulfilment of this objective. The failure to meet this objective is by far the biggest lesson learned during the development of this project: that it is important to consider the impact of all design designs and how they will propagate through the system and not just how they will simplify the immediate problem at hand. It is also by far the most significant flaw which must be addressed if the system was ever to become production quality.

3.1.5 User Interface

To meet this objective the entire system must be easy to use and only basic computer literacy necessary from the user.

The system uses standard web browser input types, in predictable, easy to understand ways. The system explains at each stage what options are available to the user and makes it clear how those options can be accessed. The question forms are clear, unambiguous and ensure that the user is only ever able to submit sensible, relevant answers. All inputs on the forms are of the type found on many web sites and will be familiar to anybody that has used the web previously. Even so, the interface is so straightforward and intuitive that even those that have little or no experience using web-based forms will have no problems successfully accessing the system and submitting forms.

It is difficult to qualify this criterion because finding someone in the University that isn’t at all computer literate is very difficult, and obviously impossible within the School of Computer Science. However, a number of people were shown the forms and asked to comment on the ease of use, and everyone asked agreed that it could not be made simpler without becoming tedious to use for those that are experienced users.

As such it is fair to say that this objective was successfully met and this part of the system completed satisfactorily.

3.1.6 Efficient

This objective required that general access to the system be as quick as possible and use only reasonable amounts of memory for an application of this type. Secondly, SQL queries and updates run on the database must complete quickly even as the database grows in size.

The general speed of the system is governed by a number of parameters: the speed of the server hardware, the server software used, the server load (the number of connected users) and the quality of the application written i.e. Acquire.

The hardware speed is beyond the developer's control as the hardware is the customer's responsibility but it is obvious that the faster the hardware used, the faster the system can potentially run.

The server software can have a noticeable affect on the overall speed. Acquire was developed using Tomcat 4.1 from the Apache Group. This Servlet container was used because it is the reference implementation of the Servlet 2.3 and JSP 1.2 standards and is required to implement both to the exact specifications laid down in the standards document. By ensuring the system works perfectly on Tomcat, it is possible to ensure that the system will work perfectly on any other J2EE Servlet/JSP container; if it fails to work on another container then that is because that container has not properly implemented the standards. Tomcat is well respected as a Servlet/JSP container and is used in many production web sites but there are many other Servlet/JSP containers available. In order for a Servlet/JSP container to qualify as J2EE is must adhere to certain standards governing the structure of web applications such as Acquire and the associated configuration files. This means that a web application designed on container can be dropped straight into another container and expected to work seamlessly. Unusually for such systems in the world of computing, this one actually works so if the customer is not happy with the performance of the Tomcat server then they are free to choose and install any other J2EE server, drop the Acquire directory straight into the new server and it will work.

As for the quality of the application software, the developer is confident that the software will run very quickly and efficiently due to the design of the software. Numerous textbooks were consulted regarding the development of such systems and how to ensure the system runs efficiently. Servlet and JSP programming has an unusual process and thread model which has to be understood thoroughly if efficient designs are to be achieved, and significant time was spent by the developer ensuring that this model was well understood and that all development embraced best practices for this type of system. It is difficult to quantify this efficiency without the system having been put through widespread use by large numbers of people, but the system did give good results under stress testing as detailed in the testing summary section of this document later.

The second efficiency consideration was that of the database access and updates. This too is governed by a number of parameters, the server hardware in much the same way as for the Servlet/JSP container, the DBMS (MySQL in this case), the JDBC driver, and the database design itself.

MySQL is widely respected as a very fast database manager, frequently recording better benchmarks than other commercial database managers such as Oracle and DB2. MySQL does lack a lot of functionality present in commercial database managers, but nothing that was required by the Acquire system so the speed advantages of using MySQL help improve efficiency without sacrificing function.

The JDBC driver acts as the bridge between the system and the database itself. The JDBC driver used is J/Connector, the default driver provided by MySQL for use with its databases. Other drivers exist which claim to offer speed improvements but these cannot be verified, nor in many cases do they support the full range of facilities provided by the JDBC API. As this development was to be as standards compliant as possible it was decided that the most commonly used and standards compliant version of the JDBC driver would be used, hence J/Connector. Testing did not show any speed problems with this driver and it is in widespread use in production class systems around the world so there should be no efficiency problems caused as a result of its use.

The JDBC driver used has good support for database connection pooling via a JNDI interface. Acquire makes full use of this connection pooling support which greatly increases the speed and efficiency of database access as connections can be reused instead of having to create a new connection each time database access is required – a process which carries very large overheads in terms of both time and resources.

The database design is very simple. It consists of a single table which contains all of the data. Initially this seems very inefficient because the main purpose of relational databases is to increase speed and flexibility by normalizing data into a number of related tables. However, the actual data stored by the database does not lend itself well to normalization. Considerable time was spent trying to design a more efficient database structure and experienced database designers were consulted, none of whom could design a better structure given the data to be stored. Also, nearly all database actions performed by the system are simply INSERTs at the bottom of the table and for this type of function the single table model is probably the quickest.

If the changes to the system discussed in the above section “Customisability” were to be implemented then the database structure would change radically and the database would normalize logically into certain tables, such as a table for users, a table for modules and a table for answers, but in the current system there are no advantages to splitting the table up. Doing so would be to force complexity into a system for the sake of it, and not because it is beneficial.

It is believed that the system is efficient in each area within its control: the application design and the database design. However, without placing the system into widespread use for a reasonable period of time and monitoring efficiency is impossible to prove that the system is efficient. Even so, confidence in the design and the results of stress testing to be discussed later mean that this objective is considered fulfilled until proved otherwise.

3.1.7 Security

This objective required that user’s passwords be securely used and not made available to any other users.

Due to the design of the authentication system, the Acquire system only knows a user's password very briefly after they login, and once the user has been properly authenticated the password is no longer required and the variable storing the password is erased. At no time is the password stored anywhere other than in memory, nor is it ever logged anywhere. If the Acquire system was ever to be placed into production it would never authenticate users itself, but would rather use an existing authentication mechanism already in place and used by the MMS system within the School of Computer Science. In such a system, Acquire would simply pass the username and password to the authentication mechanism and would get back an object describing this user and their privileges at which time the password is forgotten. As a result, it is impossible for Acquire to divulge users' passwords either accidentally or deliberately as it simply doesn't know them after the first few seconds of login.

The authentication mechanism present in Acquire currently is very weak and could not possibly be used in a production system. It was hoped that the developer would gain access to the MMS user database and write the code required to perform real authentication using that database, however once security concerns had been dealt with and the necessary code finally delivered to the developer it was too late to be incorporated into the Acquire system. However, as agreed with the customer, the system uses an Interface to describe the authentication mechanism and so any module which implements this interface can be used to perform authentication as long as it returns a specific type of user object. This means that someone could write a single new class which implements the Interface and access the real-world user database, gets details about the user and returns the appropriate type of user object. All of this would be transparent to the system as it has no knowledge of where the user credentials come from.

Given that it is impossible for the system to reveal user's passwords, this objective has been successfully fulfilled.

3.1.8 Maintainable

In order to fulfil this objective is must be possible for a future programmer to fix bugs, change and/or add functionality to the system without having to rewrite unrelated parts of the system.

The entire system was designed from the ground up to be as modular as possible. Every module has a specific purpose and performs only that purpose. Although a number of the modules are dependent on others, they are only dependent on the interfaces used and not the implementation details behind them. As long as an edited module continues to input and output data of the correct type, the exact implementation will not affect other modules.

The only section of the program that would require a rewrite is the authentication mechanism as described in the previous section. Even so, the system does not need to know what the new implementation does, simply that it deals with the correct input and output types.

Due to the modular nature of the system and the loose ties between modules, this objective is considered fulfilled.

3.1.9 Scalable

This objective required that the system must scale well if the number of students and/or modules increases over time.

From the outset the system was designed to handle heavy workloads. As discussed earlier in the Efficiency section, the system used the advanced process and thread model of Servlets and JSPs to ensure the system works as efficiently as possible regardless of the current server load. This remains true as the total number of users grows as it is the responsibility of the server to generate new Servlet threads as demand dictates as long as the system is correctly implemented to work in that way, which Acquire is.

The MySQL database is capable of storing very large databases consisting of tables up to 4 Terabytes in size which is far more than the Acquire system would ever need so it is fair to assume that the database would scale well. This coupled with the support for connection pooling means that there is no reason to suspect that the system will not scale well when required.

Although it is impossible to prove that a system will scale well until the scaling becomes necessary, every part of the system was designed with future scalability in mind. As a result, this objective is considered fulfilled until proved otherwise.

3.2 Evaluation against related work by others

To the best of the author's knowledge, there have been no similar projects this year or in previous years with which this project can be sensibly compared.

3.3 Evaluation against similar work in the public domain

There are no known research papers covering this subject so they cannot be discussed here.

There are many systems in use on the world-wide web which perform a similar role to the Acquire system, though none that perform the exact same task as it. There are countless web-based forms in use all of which are built on similar technology and principles to Acquire although they may use other server-side technologies such as PHP, CGI etc.

Compared to these other systems Acquire performs well although in certain respects it is clear that Acquire is a prototype whereas the others a production systems. Acquire concentrates on function and lacks a lot of fancy formatting which is indicative of the developers computer science background as opposed to a design oriented background. As this is a computer science project though this is no bad thing. If the system was to go into use then it probably would benefit from some "beautifying" but as it stands it performs the functions it is supposed to perform and delivers the results in a clear, concise manner.

Many on-line system use GET headers to transfer form based information across the Internet which is what causes the long, difficult to understand URLs frequently seen in the address bar of a browser when accessing such a dynamic site. However, there is nothing to stop a user manually changing the values found in the URL string which can lead to unpredictable results and also can sometimes reveal sensitive information. Acquire uses POST operations for all form submissions which means that all form data is transmitted inside the body of the HTTP request invisible to the user. This ensures that the data received by the server is as expected and it is not possible to accidentally reveal sensitive information.

A more direct comparison of systems is not possible because there is no other known system similar to Acquire and to discuss the subject further would simply to offer a general discussion of dynamic web-programming, which is not the intended subject of this document.

4 Conclusions

Overall I believe the project largely achieved what it set out to achieve. It allows users to reliably, easily and accurately submit module reviews and it allows staff members to view the results of those reviews in an equally easy, accurate and clear way.

If I could restart the project then the only area that I would completely rewrite is the form generation section, which although it generates usable forms which ask all of the questions asked on the current forms, it makes is unacceptably difficult to change the questions asked and the overall structure of the question forms. This is a severe limitation and one which single-handedly rules the system out from ever going into widespread use without serious changes.

It is also regrettable that I was unable to incorporate the real world authentication mechanism. This was caused by understandable concerns about giving a student access to the authentication database and associated source code. By the time these concerns had been settled it was too late to incorporate the authentication mechanism into the final project. I had contacted the relevant people much earlier in development and was sent a number of files and told that they were all I required. However, weeks

later when I started to code the authentication mechanism it became clear that these files were only a small part of what was actually required and that I would need extra files which provided access to much more sensitive information. This caused the delay and I was unable to integrate the real authentication mechanism, although it would be trivial for someone that understood the existing authentication mechanism (I don't) to write the authentication module as discussed earlier in the document and drop it into the Acquire system, thus gaining real world authentication.

I do not feel I can justifiably deem the project to be a success because of the failing to meet one of the nine objectives fully. The system certainly achieves its aim of proving that such a system could be implemented and made to work and that it would offer significant benefits over the existing system while having no significant drawbacks, but the inability to change the question forms rules out the system ever being used to perform its intended task in the real world unless that entire section of the system was rewritten.

5 Appendices

5.1 Project Objectives

The objectives of this project, in approximate order of importance are:

- Data collection – The system must allow non-ambiguous, accurate results to be submitted and stored in a usable way.
- Output – The system must produce output that is more reliable than the current system. It should still be possible for module coordinators to access the raw data so that manual calculations can be performed to confirm the results returned by the system are correct.
- Anonymity – Anonymity of the student submissions must be maintained at all times. It should be impractical for a module coordinator or interested party to learn the origin of a particular submission. The system administrator will be able to access the details which – while not ideal – is a necessary condition if the database is to be maintainable.
- Customisable – It must be easy to change the data regarding module details and questions.
- User-interface – Only the most basic computer literacy must be assumed of all users (both students and staff). Therefore, the user-interface must be intuitive and simple to use.
- Efficient – The database could grow to a considerable size so all operations and queries must be efficient in terms of both speed and memory usage.
- Security – As the system will use username and password combinations, it should do so securely, ensuring that one user cannot discover another user's password. Only certain people should be able to access the results of the system - primarily the module coordinators, but not the individual module lecturers.
- Maintainable – It should be possible for a future programmer to fix any bugs found in the system, change how the system works, and add new functionality without having to rewrite other parts of the system.
- Scalable – The system should be able to grow as student numbers and/or module numbers grow.

5.2 Requirements Specification – Version 1.0

5.2.1 Preface

5.2.1.1 Product Name: Acquire

This requirements specification document pertains to the development of a system to allow the submission and storage class questionnaires and the manipulation, analysis and display of results in an Internet-based environment.

5.2.1.2 Version History

Version 1.0 – First release version of the requirements specification document. Changes since version 0.1 include clarifying system diagram and "Department of Computer Science" is now referred to as "School of Computer Science".
No changes to the described system were necessary.

Version 0.1 – First requirements specification document. Version number 0.1 was chosen to indicate that this is a draft document subject to change and correction; the first release document will have version 1.0.

5.2.1.3 Intended Audiences

The intended audiences for this requirements specification documents include the following:

- Client – Dr Roy Dyckhoff

  Dr Dyckhoff should read this document to learn about the proposed system and to verify that it meets the requirements set out for the project.

- System Engineer

  The system engineer should use this document to understand what systems need to be developed for the successful completion of the project.

- System Test Engineer

  The system test engineer should use this document to development validation tests for the system.

- System Maintenance Engineers

The system maintenance engineers should use this document to understand the system as a whole, and the relationships between the individual parts of the system.

5.2.2 User Requirements Definition

5.2.2.1 Web-based questionnaires

The system will provide a method by which students can access copies of the class questionnaires via any mainstream web browser, such as Internet Explorer, Netscape, Mozilla, and Opera or any other browser conforming to recent web standards. The operating system being used will not affect the way the system works.

The questionnaires will contain a number of questions relating to the particular module being rated, an area in which the student can submit comments, and a button which allows the responses to be stored in the system for later use.

The questionnaires will be consistent in their presentation, following a fixed pattern, although the actual questions asked will vary from one module to another. The questions will be presented in a form similar - but not necessarily identical – to the following:

The system will only assume the most basic computer literacy of the user, so the interface must be intuitive and very easy to use. It will use standard user interface layouts to ensure the system appears familiar when used.

Please see the Appendix A for an example of a current form (paper version only).

5.2.2.2 Authentication

Students will authenticate themselves to the system by supplying a username and password. This will allow the system to provide access to the forms for only the modules in which that student is enrolled, thus stopping people from submitting forms

for modules they have not studied (a problem with the current system). By authenticating with the system, it also allows students to access their previous form submissions and change the values they submitted. This will only be allowed up until a certain date, at which time the data will become frozen.

Module coordinators will also authenticate themselves to the system by supplying a username and password. This will allow module coordinators to edit module question details and to view the results generated by the system (both described below).

All passwords will be encrypted before being transmitted across the network to ensure the security of the system.

5.2.2.3 Anonymity

The entire process of submitting forms should be as anonymous as is practicable. It is a necessary condition of the system that users authenticate themselves to it, which does mean that the system is not 100% anonymous. However, there should be no practical means by which a module coordinator or other interested party can access the details regarding which student submitted which entries. It should also be made clear to students that they should not include their name in the comments box at the bottom of the form, as this clearly means that the form is not anonymous.

The only person that should be able to learn which students submitted which forms is the system administrator. It is necessary that the administrator have total control over the system if it is to be maintained properly, therefore it will be possible for the administrator to interrogate the database and learn student identities. However, there will be no accessible interface which provides this information, which means that no-one else will be able to access it.

5.2.2.4 Questionnaire Contents

Each module will have its own set of questions which are displayed to the students. There will a web-based interface which allows module coordinators to create new modules and associated questions, edit existing module questions, and delete questions associated with modules which are no longer relevant. This interface will be

simple to use and will expect nothing more than basic computer literacy from the user.

Once submitted the module question details will be stored in specially formatted files and will be identified and read automatically by the system such that they are immediately accessible by the students with no further action from the module coordinator. No users will need to know details of the internal structure of the formatted files and they will never need to be edited directly.

5.2.2.5 Output

The system will generate the same output as the existing system, but in a revised format to remove some areas of potential confusion present in the current system. Initially the output will be displayed on the screen but there will also be a facility to generate printed copies of the reports.

Only module coordinators and certain other staff will be able to view the results of the system, once they have authenticated themselves to it. No-one else will be able to view the results, except at the discretion of the module coordinator, which is beyond the control of the system.

It must be possible for module coordinators to access the raw data contained within the system so that the result generated by the system can be checked for correctness. However, at no time should this breach the anonymity condition, which, together with generating reliable, accurate results, is the most important aspect of the system.

Please see the appendices for an example of the output currently produced.

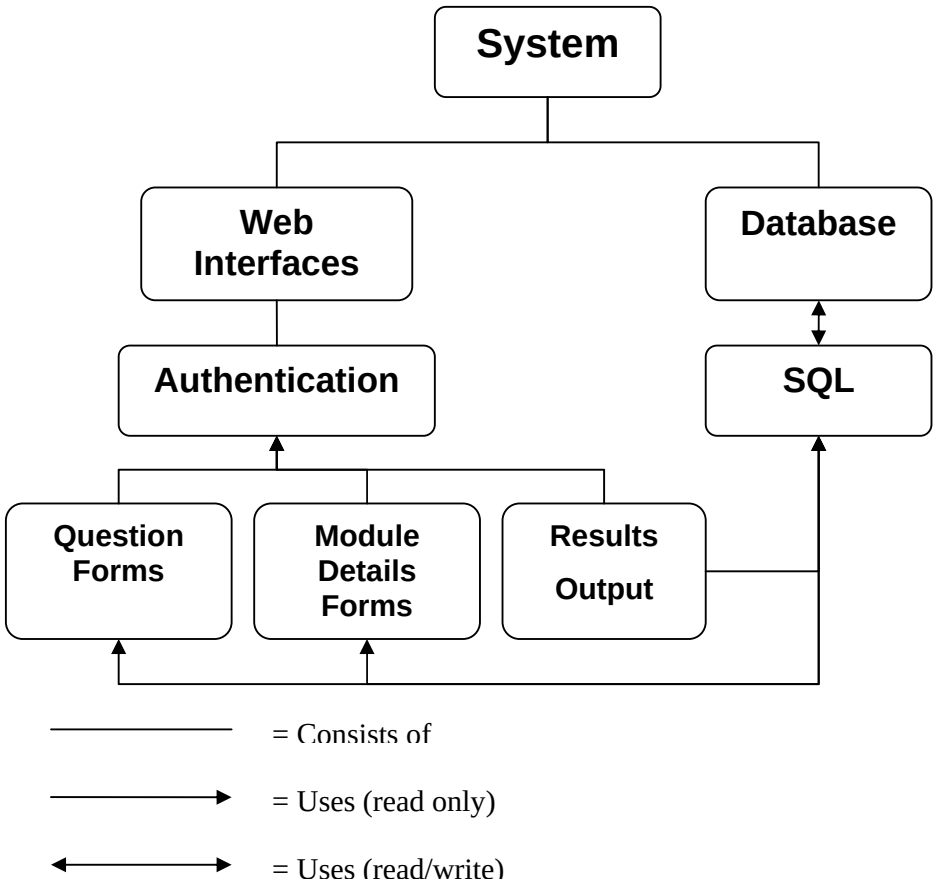
5.2.2.6 Help Details

The system should be so easy to use that no help information is required. However, basic instructions will be available in case some users aren’t sure what to do.

The form generation interface, as used by module coordinators, will contain a tutorial explaining its usage.

5.2.3 System Architecture

The system will consist of four main areas: the database, the web interfaces, the authentication layer and the SQL layer. These are connected as shown in the diagram below:



5.2.3.1 Web Interfaces

This section of the system contains the three types of interface available to the user depending upon their desired action and access privileges. Each interface will make use of the authentication layer to establish which services are available to user and to determine which module forms to display.

5.2.3.1.1 Authentication

This layer connects to the appropriate authentication database via an LDAP (Lightweight Directory Access Protocol – RFC 2251) server to gain access details for the user based on the supplied username and password.

Note: Security concerns regarding access to the live databases during testing may require this section to be revised. Possibilities are to use an LDAP server containing dummy data so that the final system can be plugged straight into the live system, or to use an htaccess access file. However, this would be unsatisfactory as it would mean the system would require new sections to be written before it could go live.

5.2.3.1.1.1 Question Forms

The forms presented to the students allowing them to rate the modules for which they are enrolled. These forms will store their results in the database once submitted.

5.2.3.1.1.2 Module Details Forms

The forms presented to module coordinators allowing them to create, modify and delete details regarding modules. Once submitted, the results will be stored in formatted files for use by the system.

5.2.3.1.1.3 Results Output

A set of read-only forms which will display the data contained within the database and the results of analysis performed on the data.

5.2.3.2 Database

A relational database will store all the form results submitted by the students for the individual modules. It must be a fast, efficient database allowing a wide range of operations (updates and queries) to be run against it. It must be possible to back-up the database easily.

5.2.3.3 SQL

The SQL (Structured Query Language) layer provides a method of interaction between the web-based interfaces and the relational database. It will allow a number of predefined queries and operations to be performed on the database. Module coordinators will be able to submit their own queries on the database using SQL to allow full analysis of the data. However, there should be mechanisms in place to ensure that data regarding student identities is still not accessible. Furthermore, these queries will only be available once all forms have been submitted and basic analysis completed. This restriction is in place because a badly crafted query could seriously

impact database performance which would not be acceptable at times when transaction numbers are high.

5.2.4 System Requirements Definition

5.2.4.1 System Implementation

The system should be implemented using a combination of server-side scripting technologies such as Java servlets, JSP (Java Server Pages), PHP (PHP: Hypertext Processor), ASP (Active Server Pages) etc, and a relational database management system such as Oracle, MySQL or SQL Server. Trials should be performed to establish the best combination for the system, based on flexibility, performance, compatibility, stability and the availability of the software within the school.

5.2.4.2 Reuse

Component reuse within the system will be minimal. The only pre-existing sections of the systems are the authentication databases (if these are to be used), but these will be accessed via a protocol rather than by assimilation into the system.

5.3 Design and Implementation

5.3.1 Development Methods

5.3.1.1 Process Model

The system will be developed using the evolutionary process model. By designing the system according to an evolutionary model it allows each module to be written independently of each other module and then plugged together using predefined interfaces. This means that for the first prototype of the system, the database element may simply be a flat-file database which conforms to an established interface understood by the database connectivity layer. By doing this it is possible to develop and test each section of the system individually as development progresses, and over time each section will be fully developed to release standard. Furthermore, this process model is also ideally suited to rescaling software projects, so this model is an excellent choice given the project objective that the system should be scalable in the future.

In addition, by adhering to this process model, if in the future another developer wishes to extend the system further it will be a relatively simple process, thus satisfying the maintainability objective of this project.

5.3.1.2 Implementation Tools & Languages

The system can be split into two distinct sections: web-based front-end and the database driven back-end. There is also a third area which doesn't belong to either section, but which acts as a bridge between the two. Each section will require a separate set of technologies.

5.3.1.2.1 Web-based Front-end

The front-end of the system (the sections presented to users) will be written using a variety of technologies, including HTML (Hypertext Mark-up Language), Java Servlets and JSP (JavaServer Pages).

The web-server hosting the system will need to be running a suitable Servlet container and JSP interpreter. The server chosen for this job is Apache Tomcat. This server was chosen because it is the official reference implementation for Servlet 2.3 and JavaServer Pages 1.2 technologies (the latest official standards) so any system written for it should work seamlessly on any other server. Tomcat also has other advantages such as being available for free; is available for nearly every platform, and as it is open-source there is a wealth of documentation available on the Internet.

All passwords must be encrypted before being transmitted across the network. This encryption must be performed using an industry standard algorithm other than DES (Data Encryption Standard), as DES is no longer considered secure.

5.3.1.2.2 Back-end Database

The software will use a scalable, multi-user relational database which will store in a usable way all the data submitted by students. For this purpose, Oracle9i Database will be used. Oracle is an industry leader in relational database technology and is noted for its stability, reliability and efficiency. It is also available for any platform likely to be used within the school i.e. Microsoft Windows, Linux and Solaris, as well as many others.

However, there may be licensing issues regarding the use of Oracle9i for this project. If that is the case, then Oracle will be substituted with MySQL, a freely available, open-source relational database system with many of the facilities offered by Oracle and good standards of reliability and efficiency.

5.3.1.2.3 JDBC

JDBC acts as the bridge between the front and back-ends of the system. It is a freely available API (Application Program Interface) which allows Servlets to connect to and use any database system that uses SQL (Structured Query Language), in a standard way regardless of the particular database software being used. By using JDBC it means that the front-end of the system can be designed and written totally independently of the back-end system as all database calls will be the same whether the system finally uses Oracle or MySQL. Furthermore, this independence is essential if the evolutionary process model is to be used.

Note: JDBC is not an acronym, though it is frequently and incorrectly assumed to mean Java DataBase Connectivity.

5.3.2 Project Management

5.3.2.1 Change Management

As the system is designed to automate an already well established manual system, it is not anticipated that there will be any fundamental changes during the course of the project. However, as there are some known changes to be made to the existing system in order to remove some inconsistencies and potential confusion, it is possible that some further changes will be requested. In this event, the following steps will be taken:

1. The client will provide a detailed explanation of the proposed change.
2. The change will be considered by the developer to establish which of the following is true:
    a. The change can be incorporated without affecting the project schedule

b. The change can be incorporated but the project may take slightly longer to complete, or require small sacrifices to be made in non-essential areas of the system if it is to be completed on time
c. The change can be incorporated but the project will take significantly longer to complete
d. The change will require a complete restructuring of the system and will increase the time taken to complete enormously
e. The change would jeopardise the completion of the project and could require the entire project to be restarted.

3. The result of the previous study will be presented to the client who will decide whether or not to proceed with the change. As the deadline for delivery of the system is non-negotiable, any result other than (a) or (b) will mean that the system cannot incorporate the change, unless the client agrees to change the project significantly. If the client decides to change the project significantly then it will be agreed that the goal of this project will shift from delivery a complete system to that of delivering a partial system which can be developed further at a later time.
4. If the change to the system is significant enough to warrant the shift of the project goal, then a new requirements document will be written and the new system developed according to the new requirements.

5.3.2.2 Version Control

The project will not make use of any concurrent version control software such as CVS (Concurrent Versions System). This decision was made because there is only one developer working on the project so version issues should not arise.

A single user version control system such as RCS (Revision Control System) may be used to keep track of the progressing system builds but this is at the developer's discretion.

5.3.2.3 Deadlines and Deliverables

Deliverable	Deadline	Status
Project Description	11/10/2002	Completed
Project Specification	30/10/2002	Completed

Project Plan	30/10/2002	Completed
Interim Report 1	4/12/2002	Completed
Interim Report 2	12/03/2003	Completed
Project Report	23/04/2003	Completed
Software	23/04/2003	Completed
Documentation	23/04/2003	Completed
Presentation	12/05/2003	Pending

5.3.2.4 Milestones

The milestones for this project are:

1. System to generate well-formed XML (Extensible Mark-up Language) files describing questionnaires from a web-based form usable by non-technical user.
2. System to correctly format and display questionnaires defined by XML files.
3. Secure authentication of users, correctly determining appropriate privileges and viewable forms.
4. Basic database operations (simple queries) successfully implemented.
5. Advanced database operations (record creation, modification, deletion etc) successfully implemented.
6. Correct data analysis performed and correct output produced.
7. System tested and shown to be correct
8. All deliverables completed and submitted on time.

5.3.3 Resources

5.3.3.1 Hardware

The project has the following available for use:

- A LAN (Local Area Network) connecting PCs and Apple computers
- Networked PCs running Red Hat Linux 7.3
- Networked PCs running Windows 2000/XP
- Networked PCs running SuSE Linux 8.1
- Networked Apple computers running MacOS X

The system will be developed using a combination of machines running Linux and Windows XP. The server and database will be initially developed on a Windows XP machine before being moved to a Linux machine for deployment.

5.3.3.2 Software

Below is a list of the software used to write, compile and test the system, but does not include any software packages which may be integrated into the final system itself such as pre-written LDAP (Lightweight Directory Access Protocol) authentication routines, data compression routines etc.

The software available is as follows:

- Apache Tomcat – The servlet container used to compile and run servlets and JavaServer Pages (this also acts as standard web server)
- Macromedia Dreamweaver MX – Used to design and code the Servlet and JSP files.
- TextPad 4 – A text and hexadecimal editor which provides syntax highlighting for Java (the language used to write Servlets and JavaServer Pages) and XML.
- Oracle 9i – Database Management System used to manage the back-end database of the system.
- MySQL – Database Management System to be used in the event that Oracle licensing issues cannot be resolved.

5.3.3.3 Resource Constraints

5.3.3.3.1 Personnel

The number of people working on the system is limited to one. If the developer falls ill or is otherwise unable to work on the project there will be no option but to postpone system development. This is discussed further in the "Risks and Fall-back Plans" section of this document.

5.3.3.3.2 Time

The system must be delivered by 23rd April 2003, which amounts to 23 weeks of development time, with no scope for overrun. If it appears that the project is going to

overrun then the steps detailed in the “Risks and Fall-back Plans” section of this document will be followed to resolve the problem.

5.3.4 Risks and Fall-back Plans

The identified for this project are as follows:

- Requirements change: The client may request changes to the requirements of the system, although this is unlikely. If this does occur, then the impact of the requested changes on the project will be assessed and the best course of action determined. It will then be up to the client to decide how to proceed. This is discussed in more detail in the “Change Management” section of this document.
- Project overrun: The deadline for this project is 23rd April 2003 and this is a non-negotiable deadline – the system must be delivered by this date. If it seems likely that the project will not be completed by this date then two options are available:
    - The project can be scaled down enough to bring it back within the time available;
    - The system will be delivered only partially, but in such a way that it can be easily completed by a future developer. This would only be done as an absolute last resort.
- Developer illness/absence: As there is only one developer working on the project, it is very dependent on that developer being able to work. If the developer is unable to work for any reason then the only option available will be to postpone the project until such a time as he is able to work again. If this occurs, then the action taken would be the same as for project overrun as described above.

5.3.5 Quality Control

To ensure that the quality of the system is as high as possible, the following quality management and review techniques will be employed:

- Thorough testing: The system will undergo extensive testing to ensure that every part of the system works correctly, and that the system as a whole works correctly. Testing procedures are detailed in the Test Plan section of this document.

- Quality review: Reviews of both the program code and documentation will be conducted regularly during the development process to ensure consistency and correctness.

  All project documents will be proofread by an independent party to make sure they are as readable and understandable as possible. All documents will be generated using templates to ensure consistency of appearance throughout.
- Due to the nature of Servlet and JSP based systems, traditional complexity measures are not easily applied and so will not be used extensively with this project. The only traditional metric which can be applied to the system as a whole is the Mean Time Between Failures (MTBF) metric, which for this system should be in the region of 3-6 months, though it is difficult to be accurate because the most likely point of failure is the Servlet container or the database management system, both which are outside the control of the system.

5.4 Testing

5.4.1 Test Plan

5.4.1.1 Black-box testing

Due to the nature of the system being designed, the majority of testing will be done at the development stage, using black-box testing. In many conventional systems, potential inputs from users are varied and often unpredictable, but in this system all possible results will have been predetermined by the questionnaires they are required to submit. As these questionnaires are generated by the system, the potential inputs are well-known in advance. Therefore, black-box testing will be used to test the questionnaires as the inputs are known, and the generated outputs can be monitored in the database. If the outputs match the inputs, then it can be assumed that that section of the system is correct.

5.4.1.2 Modular testing

The section of the system which allows users to generate, edit and delete question forms will be tested using module test methods. This section of the system will be

tested to ensure that regardless of what input is given by the user, the XML files generated to define the questionnaire are always well-formed and readable by the form display modules in such a way that guarantees the form will be rendered correctly every time.

5.4.1.3 Testing the database

During development of the database operations routines, each operation will be tested extensively to ensure that it operates efficiently, quickly, safely and always returns the correct results. All queries will be tested on datasets of varying size to ensure that they are scalable from small datasets to very large datasets. Also, extreme care will be taken to make sure no outer-joins are runnable which can seriously affect the performance of the database, and the system as a whole.

5.4.1.4 Stress testing

As this system is designed to run on a web server being accessed from many different clients at a time, it is essential that the system is shown to be capable of dealing with many transactions at once. If the system is rolled out to each school the number of clients connecting to the system to submit module reviews could be as high as 5,000 over a relatively short period of time, although this is very unlikely.

Therefore, the system must be tested with multiple clients together, and shown to be able to handle the stress adequately, in both the main web server dealing with requests, and the database performing the operations. It is not possible for 5,000 clients to all test the system at once, but the system will be tested with as many clients as possible. Whether this will be an automated process, or whether other people within the school will be enlisted is undecided, but it will be by the time the testing is necessary.

5.4.1.5 Security Testing

A primary concern of the system is that all forms submitted should be done so anonymously, that is, it should be impractical for anyone to learn which student submitted which form. Therefore, significant testing will be done to ensure that all submissions are anonymous.

The developer will conduct the initial testing, using their knowledge of the internals of the system to guarantee that the data is anonymous. Once the developer is satisfied, they will enlist the help of other members of the School of Computer Science, and various other people with experience in this area, and ask them to try and discover which student submitted which form, using all reasonable means. If none of these people succeeds in learning the identity of a student, then the system will be considered anonymous.

Note: During this phase of testing, testers will be instructed not to do anything which jeopardises the security or stability of any system other than that being tested. Only people trusted by the developer will be asked to perform this type of testing.

5.5 Project Monitoring Sheet

Task No.	Task Name	Duration (Weeks)	Dependencies	Completed
T1	Requirements specification	1	-	✓
T2	Project Plan/Context Survey	2	T1	✓
T3	XML Format Definition	1	-	✓
T4	Coding: XML Form Generation Interface	1	T3	✓
T5	Coding: Questionnaires Generated From XML	1	T3,T4	✓
T6	Coding: Authentication	2	-	✓
T7	Coding: Simple Database Connectivity	2	T6	✓
T8	Coding: Simple queries/operations	2	T7	✓
T9	Coding: Advanced Database Connectivity	2	T6,T7,T8	✓
T10	Coding: Advanced queries/operations	2	T9	✓
T11	Coding: Data analysis and output	2	T10	✓
T12	Coding: Online user manuals	1	T4,T5	✓
T13	Product Testing:	4	T3,T4,T5,T6, T7,T8,T9,T10, T11	✓
T14	Documentation	3	T2	✓
T15	Project Report	4	T1,T2,T14	✓

5.6 Interim Report 1

5.6.1 Design Decisions

At the time the project specification and plan was written it was still not known whether it would be possible to use Oracle for the database or whether the free alternative, MySQL, would have to be used instead. Since then it has been confirmed that the university has a license for Oracle so this will be used. However, it is not currently known which version of Oracle the university has a license for. Andy Robinson has been e-mailed about this but he is yet to reply. Development has not reached a stage where this is a problem and will not do so until late February 2003 when initial testing of the system will be performed in the labs to replicate the eventual deployment environment. Andy will be e-mailed again shortly and if he fails to reply again I will seek him in person.

5.6.2 Schedule

According to the project monitoring sheet submitted as part of the project plan, development is already behind schedule. However, it was anticipated that development would be difficult during the last few weeks of term when deadlines for taught modules invariably begin to accumulate. As a result, certain tasks early in development were listed as being of one week duration when in actuality they will not require that long to complete, therefore they will be easily completed during the Christmas vacation such that the project will be back on schedule by the end of December 2002.

This revised schedule does take into account preparation for the January exams.

5.6.3 Other Notes

The product name has been changed to “Acquire” – “Automated Class Questionnaires Using Internet Related Equipment”.

Although horribly contrived, it seems more appropriate than the original name of simply “ACQ”, which didn’t mean anything.

5.7 Interim Report 2

5.7.1 User Authentication

In a recent meeting with the project supervisor the largest problem facing the project was discussed: how to authenticate users and retrieve the list of modules on which they are enrolled.

Following an e-mail discussion with Ross Nicoll this problem has now been solved. Ross explained how the current system works and was kind enough to send the actual authentication source code used by the department's existing web-based systems such as MMS which can be integrated into this system with only minor modifications. This means that there are no problems with my system using the real authentication server but initially this won't be used as that would restrict testing to one account only (my own) as obviously this is the only account for which I know the details.

5.7.2 Design Changes

Database: The back-end database software has been changed from Oracle 9i as originally specified and will now use MySQL. This change was made because the setup and administration of the Oracle database was too time consuming for a project of this size.

Design environment: The system is now being developed using a combination of JavaServer Pages (JSPs) and Java servlets instead of using servlets exclusively as originally stated. This change increases the separation between back-end systems and the user-interface meaning the system adheres more closely to the model-view-controller paradigm. It also greatly simplifies maintenance of non-core areas of the system such as user-interface appearance.

5.7.3 Schedule

Development has not progressed in line with the schedule set out in the project plan but the project will still be completed well within the available time-frame. A first prototype of the system is to be demonstrated on March 20th. This will include the

user interface, form submission and the on-line form design but probably not any database interaction.

5.8 List of changes

No changes were made to the requirements document or the project plan during the course of the project.

5.9 Testing Summary

Due to the preconceived forms used by the system it was possible to know the exact inputs the system was going to receive which meant that most testing performed was black box testing. Once the module had been written a dummy HTML or JSP page would be created which could send a possible set of inputs to the Servlet or JSP page (JSP pages can be linked so that they act upon one another) and the output monitored to ensure it was correct.

As more and more modules were completed they were plugged together to form the chains of execution. The execution of Servlets and JSPs is frequently very linear in that it will call successive Servlets each of which performs the next link in the chain, for example:

Login → Form Selection → Form Display → Form Processing →Database update → Form Selection

Each of which is just acting on the input created by the previous link in chain, and the valid inputs are known at each point. Therefore, as each successive module is written it can be added to the end of the chain and the entire chain up to the point can be tested. Because the range of inputs is limited once a module has been tested and shown to act correctly on the inputs provided it can be inserted into any appropriate chain and assumed to work. If chain testing shows a fault then it is invariably to be found in the latest module to be added to the chain.

5.9.1 Testing Form Display

The question forms are displayed by applying XSLT transforms to the source XML files. As the XML files can contain only a limited number of questions, and always in a specific order testing was performed by generating every possible permutation of XML file and using the XSLT transform. By doing so it was possible to prove that every possible form would be displayed correctly regardless of the exact nature of that form.

5.9.2 Testing the Database

Database testing was simplified greatly by the very simple, one table design of the database.

The database was tested for all possible queries and operations with the database in different states, including empty, loaded with a small dataset and loaded with a large dataset.

In order to successfully pass testing it has to correctly perform all INSERTs, UPDATEs, SELECTSs and DELETEs regardless of the initial state of the database.

All such databases calls were tested both via the command-line interface into MySQL and via the JDBC API as used by the Acquire system.

The databases also underwent stress testing in conjunction with the Tomcat server using JMeter as detailed below.

5.9.3 Testing the Tomcat Server

In order to guarantee that the web server would cope under stress it was tested using a freely available tool from the Apache Group called JMeter. JMeter is a tool designed to automatically stress test a server and all the resources that server uses such as databases and to generate performance analysis of the server. As it was not feasible to organise several hundred people to all access the Acquire system, JMeter was used to simulate such a load.

During JMeter testing the server, databases and Acquire all showed that a heavy load would not adversely affect overall performance, except for a slight delay in initial response times which is to be expected of any server under load. Acquire showed that it continued to process inputs, perform database queries and generate output reliably even under heavy loads.

5.9.4 Testing the security

Because security is such an important aspect of the system but also one which is very easy to get wrong, industry standard implementations were used. The encryption of passwords is performed by the Tomcat server using 128-bit SSL encryption which has undergone lengthy scrutiny by the cryptanalysis community. Likewise, the SHA-1 implementation used for the secure hash of the username is generated using the Java Cryptography Extension pack as provided by Sun which has undergone much independent testing and so far proved secure.

As a result it was not possible to test these directly but they can be assumed to be secure, and certainly more secure than anything I would write.

5.10 Status Report

The system as delivered performs all functions required of it. It is not however, ready for real world deployment and would require certain changes to be made or sections to be modified if it was to be used widely.

The most important change needed if the system was to be deployed would be the inclusion of a secure and correct authentication mechanism, presumably based on the existing authentication procedures in use by the Module Management System (MMS) within the department. The authentication mechanism that is currently in place is a very weak file based implementation which stores user details in a comma separated file containing passwords in plain text. This is not an acceptable mechanism for deployment because it stores unencrypted passwords, but also because to be used a tool would have to written which could interrogate the MMS authentication database, extract information about each user and generate the correct entry in the text file which is clearly not a viable option.

Also, before the system is deployed it would be desirable to remove and rewrite the form generation section such that it uses a more dynamic, database driven architecture that doesn't depend on static XML files. However, this is not an insignificant change and would require rewriting large sections of code. Ideally this change would be linked to an overall design change in the database such that the users, module details,

questions and answers were all linked in various tables within a single database but this would require changing all of the code which interacts with the database, which is most of it. An intermediate option would be to leave the existing database and access as it is, but to create a second database which is used exclusively to generate the forms. By doing this it would be possible to fix the form generation deficiencies without breaking the rest of the system.

5.10.1 Major Contributions

The major contributions of this project were the successful solution to the problem of ensuring form submissions are anonymous using the secure hash generation – a feature which could be easily and successfully integrated into any future system.

Although the way in which forms are generated is unsatisfactory, the actual forms themselves are good in that they are very clear and easy to understand, solving all of the problems present in the existing paper based system.

The results display is also much clearer, more informative and easier to understand than the existing system so this could be taken forward into a future version.

5.10.2 Deficiencies

As repeatedly stated, the form generation section is not acceptable for a production system.

Also, the authentication mechanism needs to be updated before a real-world deployment.

6 Glossary

XML – Extensible Mark-up Language

XSLT – Extensible Style sheet Language for Transformations

HTML – Hypertext Mark-up Language

XHTML – XML based implementation of HTML

SQL – Structured Query Language

Xalan – The XSLT processor from the Apache Group. A Xalan is a rare musical instrument.

CSS – Cascading Style Sheets

SHA-1 – Secure Hash Algorithm 1. Successor to Secure Hash Algorithm

MAC – Message Authentication Code

NIST – National Institute of Standards and Technology

NSA – National Security Agency

W3C – World-Wide Web Consortium

PHP – PHP: Hypertext Processor

CGI – Common Gateway Interface

JDBC – Not an acronym, but commonly referred to as Java Database Connectivity

JNDI – Java Naming and Directory Interface

DBMS – Database Management System

J2EE – Java 2 Enterprise Edition

URL – Uniform Resource Locator

HTTP – Hypertext Transfer Protocol

API – Application Programmer Interface

7 References

[1] http://www.st-andrews.ac.uk/publications/univ_statistics.shtml - Facts and figures about the University of St Andrews.

[2] http://www.itl.nist.gov/fipspubs/fip180-1.htm - A technical description of SHA-1

8 Additional Appendices

[Hard-copy only]

Assessment for CS4099/98 Software Project – Revised Nov '02

This version of this document supersedes the earlier version circulated to and discussed with students.

CS4099/4098 is assessed as described below: this replaces the material on page 42 of the Handbook. CS4099 and CS4098 are assessed by the same criteria, but with the understanding that for CS4098 less can be achieved in the smaller amount of time available. This is reflected by expectations of less ambitious objectives, smaller amounts of code, and a shorter report. It is extremely important from the first stages that you are aware that less work is expected, as the objectives you agree with your supervisor help to determine the progress of your work.

The project is marked by two examiners, normally including the Project Supervisor. Presentations may be assessed by members of staff other than the supervisor and second marker.

The project is assessed on a number of "basic", "additional" and "exceptional" criteria, on a 5 point scale

E- inadequate	D- adequate	C- satisfactory
B- good	A- excellent	

Grades according to the University scheme are assigned according to thresholds as follows:

1–3	The project is inadequate in all of the basic criteria.
4–6	The project is inadequate in more than one of the basic criteria, but not all.
7–8	The project is inadequate in one of the basic criteria.
9–10	The project is adequate on the basic criteria.
11–13	The project is at least satisfactory on almost all the basic criteria and is satisfactory on most of the additional criteria.
14–16	The project is at least good on almost all the basic criteria and is at least satisfactory and sometimes good or excellent on the additional criteria.
17	The project is good or excellent on almost all the basic and additional criteria.
18–19	The project is good or excellent on all the basic and additional criteria and also has elements of the exceptional criteria.
20	The project is good or excellent on all the basic, additional and exceptional criteria.

BASIC CRITERIA

Understanding of the Problem A
Comment: Fine
Proper Software Engineering Process (including Plan) A
Comment: Fine
Achievement of main objectives[1] A-
Comment: The demo indicated there to be good achievement, but more thought could have been given to examples confirming this.
Structure and Completeness of the Report A-
Comment: The report was well written.
Structure and Completeness of Presentation B+
Comment: The presentation was very reasonably structured but again more thought could have been given to examples.

ADDITIONAL CRITERIA

Knowledge of the literature B
Comment: Not much was demanded here.
Critical evaluation of previous work B
Comment: A bit over the top at times, e.g. 'next to useless existing system'.
Critical evaluation of own work A-
Comment: Thoughtful.
Justification of design decisions B+
Comment: Good with one major failure to anticipate.
Solution of any conceptual difficulties B+
Comment: Security was well done, customizability less so.
Achievement in full of all objectives[1] B+
Comment: One major failure re customisability.
Quality of Software A-
Comment: Good overall.
Ambition and Scope of Project B+
Comment: Good overall within an inherently limited framework.

EXCEPTIONAL CRITERIA

Originality of concept, design or analysis B+
Comment: Original enough.
Adventure B+
Comment: Did what he could.
Inclusion of publishable material B+
Comment: Could be published, though this sort of area lacks publications.

•1"achievement" covers achievement of the original objectives, achievement of modified objectives or provision of convincing evidence that the objectives are unachievable.

Report on SH project CS4099 by Gareth Edwards, Summer 2003.

Criteria

Understanding B
Main problem is failure to plan
for customisability; well, he
worked that out by the end, but too late.

Proper SE A
Very satisfactory

Achievement of main objectives C
OK except for non-customisability
and proper authentication

Structure and completeness of the report A
Main report a bit on the short side.

Structure and completeness of presentation
Not known

Additional Criteria

Knowledge of the literature B
Not a lot that we could find...

Critical evaluation of previous work B
Difficult when there is so little

Critical evaluation of own work A
Has put a lot of effort into this

Justification of design decisions C

Solution of any conceptual difficulties A
I'm pleased that he achieved
what is, I think, a workable solution

to the anonymity problem

Achievement in full of all objectives B
Customisability not achieved
Proper interaction with something
like the data warehouse not achieved

Quality of Software B
No obvious way of getting good
summary printouts on A4.

Ambition and Scope of project C
Could have been more ambitious by
planning more for customisability

Exceptional Criteria

Originality of concept, design or analysis B
Supervisor's idea; student's design and
analysis.

Adventure C

Inclusion of publishable material C
Maybe I would be more generous if I
could see this being in use in a year's
time; but I can't, yet.

Overall grade 15
(assuming grade B or A on presentation--I didn't see it.)

RD

Final grade 17

An inherently limited project was done well.

After discussion with RD, **the agreed grade was 15**.

X-Ninja

Xml **N**otation **In**to **Ja**va

A conversion tool.

Chris Mannion

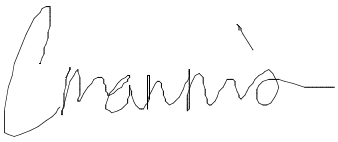
The University of St Andrews

Abstract

The purpose of this project was to create a tool that could convert types in an XML schema definition into classes and variables in the Java programming language. The tool translates structures within the XML schema, such as elements, complex type definitions etc., into Java class definitions based on a set of mapping rules. The mapping rules can be user defined/customised to allow the outputted Java classes to be tailored to a user's specific needs.

The ‘tool’ is a set of Java classes and methods that are designed to be useable either as a resource to other Java programs or with a simple GUI placed on top of them to make an application.

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.
The main text of this project is 15,813 words long, including the project specification and plan.
In submitting this project report to The University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.



Christopher J. Mannion

CONTENTS

1. Introduction
    1.1. XML: an overview of the extensible markup language.
        1.1.1. XML syntax
        1.1.2. DTD and XML Schema
            1.1.2.1. XML Schema
    1.2. Java: an overview of the Java programming language.
    1.3. DOM: an overview of the document object model.

2. Project Details
    2.1. X-Ninja's function: what the program does.
    2.2. X-Ninja's mechanism: how the program does it.
        2.2.1. translator class
        2.2.2. treeHolder class
        2.2.3. rules class
        2.2.4. codeWriter class
        2.2.5. javaSyntax class
        2.2.6. elementOps class
        2.2.7. attributeOps class
        2.2.8. simpleTypeOps class
        2.2.9. complexTypeOps class
        2.2.10. classOps class
        2.2.11. orderIndicatorOps class
        2.2.12. fileHolder/variableHolder/restrictionHolder classes
        2.2.13. listNode/myQueue classes
        2.2.14. exampleInterface class

3. Evaluation and Critical Appraisal

4. Conclusion

5. Appendices
    5.1. Appendix 1: Project objectives and plan / Interim report 1 / Interim report 2
    5.2. Appendix 2: Testing summary
    5.3. Appendix 3: Status report
    5.4. Appendix 4: Maintenance document

The purpose of this project was to create a ‘tool’ that could be used to parse an XML schema definition and translate its contents into an equivalent set of Java classes. There are many features that could be applied to a project such as this, some of which where attempted during this project, but the most basic requirement of the project was to be able to successfully parse an XML schema file and produce some corresponding output in Java code.

XML

At the simplest level eXtensible Markup Language (XML) is a powerful, generic mark-up language. XML can be used to describe and store any data in any number of ways, with no predefined mark-up the language can be infinitely adapted to tailor to the needs of any kind of data or information. XML is attempting to be a truly universal data description language, with its ability to be tailored to any data, the fact that it uses Unicode as a standard character set so that numerous writing systems and symbols are supported and the mechanisms available for checking the integrity of an XML document.

In fact, even though XML is such an open standard, it still has many rules and ways of governing data and checking that data is in the form that the user would like it to be in. Any XML document can be a ‘Well formed’ and a ‘Valid’ document. A well-formed XML document is one that contains all correct XML syntax and a valid XML document is one that conforms to a DTD or XML schema definition.

XML syntax

In XML, all data is enclosed in ‘elements’. An element is declared by a tag, which is an identifier name enclosed in angle brackets, i.e. ‘<the_element>’. The name of an element can be anything at all with a few exceptions:

- They cannot start with a numeric or punctuation character
- They cannot start with the letters ‘xml’ (with any combination of case)
- They cannot contain spaces
- They are case sensitive

Other than those rules names can contain any characters in the Unicode character set and can be of any length. However, by convention names are usually lower case and, because

they are the only things that describe the data that they hold, it is best if a element name is as descriptive as possible as to its contents.

It is said that the tag ‘<the_element>’ opens an element named the_element. Once an element has been opened like this it must be closed again somewhere later in the document for the document to be classed as well formed. The syntax to close an element is a tag containing the name of the element prefixed with a forward slash character, again encased in angle brackets, i.e. ‘</the_element>’.

The data that an element contains, its contents, are all the things that occur in the XML document between the opening tag and closing tag of the element. In some cases this could be a simple value such as in the following examples.

```
<greeting>Hello</greeting>
<int>57</int>
```

However, elements can also contain other elements. This leads to the inherent structure of an XML document, with elements able to have children (elements it contains) and a parent (the element it is contained within) element. This is what allows XML to represent strictly structured data and works similar to the way HTML allows elements to contain other elements, e.g. the <html> element will usually contain a <head> and a <body> element, which in turn contain other elements. In XML this can be done for any data, for example the contents of a letter could be stored as follows.

```
<letter attempt = “1”>
        <to>
                <name>Marc</name>
                <address>Marc’s House</address>
        </to>
        <from>
                <name>Me</name>
                <address>My House</address>
        </from>

        <message>
                <body>Hello</body>
                <ps>Bye</ps>
        </message>
</letter>
```

In this example, all of the information is part of the letter and so is enclosed in the letter element. Within the letter, the contents have been broken down further into smaller elements to make the data clearer. Another important aspect of XML is that it can be a transparent way of representing data, meaning that the data is readable by people as well as computers.

Also shown in the letter example is the second way of enclosing data within an XML element, attributes. In the opening letter tag, as well as the element name, the tag contains ‘ attempt = “1” ’. This is an attribute of the letter element, with the name ‘attempt’ and the value 1. An element can have any number of attributes, each of the form ‘ name = “value” ’ and separated by spaces. The value of each attribute must be enclosed in either single or double quote marks, either is acceptable except when the value contains single or double quotes marks and then the other form must be used. Because attributes are enclosed entirely within a tag, rather than between the open and close tags of an element, elements containing only attributes can be opened and closed in the same tag, in the form ' <the_element attribute = “value” /> ‘. While attributes are just as acceptable a way of storing data in an XML document, having an abundance of attributes instead of child elements can lead to the data looking very garbled and losing it readability to humans.

For the document to make sense both to a human reader and the computer, all the elements in the document must be correctly nested. This means that any child element must be properly closed before its parent element is closed. This is similar to identifier scope in programming languages, if an identifier (i.e. the element name) is declared within a structure (i.e. the parent element) then it doesn’t exist outside of that structure and so trying to operate on that identifier after the end of its parent structure (such as trying to close the element) is not possible.

An XML document must conform to two further rules before it can be considered well formed. Firstly, the first line of the document must be the XML declaration, defining the XML version and character encoding being used, e.g. ‘ <?xml version="1.0" encoding="ISO-8859-1"?> ‘. Secondly the document must have a root element, one within which all of the other elements in the document are contained.

DTD and XML Schema

So XML is very successful in allowing free, structured data descriptions. It can be used to describe any kind of data due to the fact that XML elements are completely defined for the user for their own purposes. However, to make XML useful on a global scale, for passing information between several users there has to be some restrictions put on what each XML document can contain. For example, it makes no sense to have elements describing a

bibliography in a document describing bank account transactions or to have a recipe for gingerbread in the middle of an inventory of car parts.

There are two mechanical ways of restricting the data that is allowed in an XML document, those are Document Type Definitions and XML Schema. On the theoretical level, both DTD and XML Schema work in the same way. They contain definitions of elements and attributes that are allowable in an XML document, their names, the type of their content and other properties such as the number of times they can occur and allowable values for content (restrictions).

XML Schema have effectively superseded DTD in the job of document description and are preferred over DTD for several reasons. Firstly, schema are written in standard XML syntax, this means that anyone who can write XML can easily write XML Schema rather than having to learn a new syntax to use DTD. Secondly, schema are extensible, allowing them to be updated and expanded on as the need arises without causing documents that conform to the old schema to become invalidated. XML Schema are also generally more powerful and allow more control over document content than DTDs, they support some base XML data types and XML namespaces and allow stricter limits on the textual contents of elements.

XML Schema

Because schema are written in XML syntax, they should conform to the rules for well-formed XML. The required root element of a schema document is always the <schema> element, which often contains attributes describing information about such things as namespaces and the schema file location.

Inside the root element, there are two different types of structures that can be defined, simple types and complex types. A simple element is one that only contains a textual value, it cannot contain any children elements or any attributes, whereas a complex type element can contain any combination of text and/or elements, or can be empty.

Simple Types:

Simple types are either a simple element or an attribute definition. It is important to remember that when one talks about an attribute definition in a schema document, the definition itself is not in the form of an attribute in the syntactical sense. Instead it is an element that describes some properties of an attribute that can be used in an XML document conforming to the schema. In the same way an element definition describes properties of an element that can be used. The most basic requirements for defining a simple element or attribute in a schema is that each should have at least a name attribute, it is also usual for

them to have a type attribute whose value should be the name of another defined element or one of the XML base data types. A simple element and an attribute can be defined most straightforwardly in the following way.

```
<xsd:element name="the_element" type="xsd:string">
<xsd:attribute name="the_attribute" type="xsd:string">
```

These two could then be used in an XML document conforming to this schema in the following way.

```
<the_element the_attribute="a string">a string</the_element>
```

There are several other attributes that can be set in an element defining elements and attributes that describe other details about the way the element or attribute being defined should be used. For both elements and attributes, fixed and default values can be set by using the 'fixed' and 'default' attributes in the definition. If the 'default' attribute is set then any element or attribute that isn't specifically set in an XML document will default to the value in the 'default' attribute. If the 'fixed' attribute is set then an XML cannot change the value, the contents of the element or value of the attribute will always be that value in its 'fixed' attribute. Attribute definitions can also include a 'use' attribute, which can be set to values of "optional" or "required" to describe whether elements that *can* have the attribute being defined *must* have it or whether it is optional.

A further complication of simple type definitions is the possibility of restrictions. A simple type with restrictions is defining a new simple type by taking an existing type and restricting the values that can be stored into it, thereby making a new type. For example, taking a number type and restricting it to only hold numbers above zero, thereby creating a positive number type. The restrictions can be of various styles depending on the type of data that will be stored in the type.
All types can have the following restrictions:

- Enumeration: The restriction element contains a series of 'enumeration' elements. The value stored in the restricted element must be the value of one of the 'enumeration' elements.
- Pattern: The value stored in the restricted element must successfully match the regular expression defined in the value of the 'pattern' element.

String based types and lists can have the following restrictions:

- Length: The value stored in the restricted element must have exactly the number of characters or list items set in the value of the 'length' element.

- Max Length: The value stored in the restricted element must have a number of characters or list items equal to or less than the value of the ‘maxLength’ element.
- Min Length: The value stored in the restricted element must have a number of characters or list items equal to or greater than the value of the ‘minLength’ element.
- White Space: The ‘whiteSpace’ restriction element can have one of a series of values that describe how white space in the value of the restricted element should be treated. These include preserving the white space exactly, replacing all white space characters with spaces or removing all white space characters and sequences of white space characters and replacing them with single spaces.

Number based types can have the following restrictions:

- Fraction Digits: The number in the restricted element can have no more decimal places than the number declared in the ‘fractionDigits’ element.
- Max Exclusive: The number in the restricted element must be smaller than that declared in the ‘maxExclusive’ element.
- Max Inclusive: The number in the restricted element must be smaller than or equal to that declared in the ‘maxInclusive’ element.
- Min Exclusive: The number in the restricted element must be larger that that declared in the ‘minExclusive’ element.
- Total Digits: The number in the restricted element must have the exact number of digits specified in the ‘totalDigits’ element.

Restrictions over a simple type element are declared in the following way.

```
<xs:element name=”restricted”>
        <xs:restriction base=”xs:string”>
                <xs:length value=”8”/>
        </xs:restriction>
</xs:element>
```

In this example the element “restricted” is defined, the defining element has a child element that is a restriction. The restriction element’s ‘base’ attribute gives the type that the restriction is based on, and therefore the type of data that this new ‘restricted’ element can contain. The restriction element has child elements defining the restrictions that are to be put in place, in this case the ‘length’ element means that any content of a “restricted” element must be exactly 8 characters long. In normal circumstances a simple type definition would consist of a single element in the schema, however the presence of restrictions is a special case that means that the simple type definition element has a series of child elements, the ‘restriction’

element itself and *its* children, the restrictions. This is important for the design of a tool to resolve type definitions in a schema. It means that tool will have to look ahead from any simple type definition it thinks it has found to check for other elements that may effect the type such as restrictions, or complex type elements as will be seen in the next section.

Complex Types:

As stated above an element is of a complex type if it is empty, if it contains other elements, if it contains only text and has attributes or if it contains a mixture of text and elements. Each different form has a specific 'tree' of elements that are used to define them in a schema. An empty element is defined as an element that could have further elements as content but has no elements defined within it. This can be done as follows.

```
<xs:element name="empty">
        <xs:complexType>
                <xs:attribute name="att" type="xs:string"/>
        </xs:complexType>
</xs:element>
```

An element containing other elements is defined in a similar way, but with further, complex or simple, elements defined or referenced inside of it. For example

```
<xs:element name="containsElements">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="subElement" type="xs:string"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

When a complex type definition has other elements used within it like this, the definition of or reference to those other elements must be contained within what is called an order indicator element. In this example, the '<xs:sequence>' element is the order indicator, which means that the elements used within it must appear in this complex type element in the exact order they are listed in this definition. The other possible order indicators are 'choice', which means that one of the child elements listed in the definition must appear, and 'all' which means that all of the child elements must appear once and only once. The order indicator element is another step in a complex type definition that a translation tool must look through to resolve the true nature of the complex type.

A complex element that contains only text and attributes is said to contain only simple content and is defined as such using the <xs:simpleContent> element as follows.

```
<xs:element name=”textOnly”>
        <xs:complexType>
                <xs:simpleContent>
                        …
                        …
                </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

If the simple content element is used in the definition it must be followed by either a restriction element and set of restrictions, or by an extension. An extension is similar in form and function to a restriction except where a restriction takes a base type and limits the content that can be put in it, an extension expands the content. An extension does not contain special extension types like a restriction has restriction types, instead the content that is to be added is just defined within the extension element as content would be in any other complex type. The difference being that the new complex type being defined will have this new content *as well as* the content of the type it is extending.

Finally a complex type can be defined that allows both complex and simple content, i.e. elements, text and attributes. This variation of a complex type is defined in a very similar way to a complex type containing only attributes except that the ‘<xs:complexType>’ element has a ‘mixed’ attribute set to the value “true”. The kind of complex type is used to allow plain text to be used in-between child elements when an element of this type is used in an XML document. This can be advantageous when a user wants data to have a more readable and natural appearance, with elements occurring inside plain text sentences for example.

It is significant to note that it is possible for the first element of a complex type definition to be similar in form to the definition of a simple type. This is not always the case, complex types can be defined using just a ‘<xs:complexType>’ element around the content (i.e. not contained within an ‘<xs:element>’ element) further complicating the job of the translation tool. However, the similarities between some simple and complex types will be important for the X-Ninja translation code as in cases like this it will have to look past the first element to explore the possibility of a complex type. This is a common theme in XML Schema, as well as there being several different ways of defining a single type, many of these ways can over lap with ways for defining other types. This has to be kept in mind when designing a tool to resolve type definitions.

That concludes the overview of structures that are available in an XML Schema. XML and its related schema present a difficult challenge for an automated translation tool. The concept of XML itself allows for a complete freedom in the way data is represented, this combined with the rigid structure types in XML Schema leads to several different valid ways of defining each type in a schema. The X-Ninja translation tool will have to deal with all of these valid cases and be able to distinguish between those variations that overlap to any extent. However, the clear structures and types defined in an XML Schema should lend themselves well to being mapped into equivalent structures and types in the Java programming language.

Java

Java is an object orientated programming language. Object orientation means that the code and data of a program is broken down into modules called objects. The overall workings of a large program can be seen as a series of 'black boxes' that interact with each other via public methods with the inner working of each of the boxes being hidden from each other. The 'black boxes' are called objects. In Java the objects are instances of things called 'classes', which are one of the inherent structures of the Java language. The Java hierarchy is quite simple, a package is a group of related classes, a class is a structure containing data and methods, data is held in variables in a class and methods are snippets of functional code. A package has no explicit declaration or code for itself, instead classes that are members of the package declare this at the start of their definition. A class however is defined, most classes in Java are defined in their own separate file, the exception being local classes that are defined in the file of and only used by another class. A straightforward class could be defined by the following code.

```
public class newClass
{
        …
        …
}
```

In between the curly braces that mark the opening and closing of the class, there is code declaring the variables and defining the methods that the class has. As well as classes that are distinct such as in the above example, classes can also extend other classes and implement interfaces. An instance of a class that extends another will automatically contain all the variables and methods that are declared and defined in the class being extended as well as those declared and defined in the extension class. The only exception to this being if the extension class uses a variable or method with the same identifier name as one in the class being extended, in this case the new variable or method will replace the old one. An

instance of a class that extends another can also be treated as being an instance of the class it extends. For example –

```
public class circle extends shape
{
        …
        …
}
```

A circle object will have all the variables and methods (except any that it has overwritten) of a shape object and could be legally treated as one. However it may also have further, circle-specific variables and methods that a basic shape object would not have.

A Java interface cannot, in itself, be instantiated and so cannot be treated as a class. However, what an interface does do is to declare the names and types of variables and the names, return types and parameters of methods that any class implementing the interface must have in it. This has an effect similar to one of those of extension, it allows instances of any classes implementing an interface to be treated as objects of the type of the interface. For example –

```
public interface letter
{
        int number;
        public int getNumber();
}

public class A implements letter
{
        …
        …
}

public class B implements letter
{
        …
        …
}
```

In this example, A and B could have completely different purposes but because both implement the ‘letter’ interface, instances of both could legally be treated as letter objects.

Both A and B must contain an integer type variable called number and a method called getNumber which takes no parameters and returns an integer value. The contents of the 'number' variable and the actual workings of the 'getNumber' method could be completely different in each but both could still be treated as letters.

You will notice that just as every XML element must be closed at the same level as it is opened, a Java class (and the same is true for methods and various other code structures) must also have a closing bracket on the same level as its opening bracket. For classes, this is usually the top level of the file they are in because everything else is contained within the class. This will be another significant point for the translation tool. If the tool is defining a class then once it has finished generating the contents of the class it must be aware that it is still within a class to be able to write the closing brace at the end of the class file.

The variables a class can contain can be of any type, like XML there are base types such as int, char and byte. Like XML also, variables can be of user defined types in that a variable can hold an object, in which case the variable is of the type of the class the object is an instance of. However, unlike XML, Java does not require different kinds of structures to hold different kinds of content, everything in Java is contained in a class. When a variable is declared in Java it is declared in the form of the type it will hold followed by it's identifier name, such as

```
String name;
byte thingy;
```

This is just the same as a simple type element being declared in a schema and being given name and type attributes. However, unlike schema elements, Java has no direct way of attaching any limits on the acceptable values of a variable (except for those inherent to the variable type).

It is important to remember that while XML is a text based data description language, Java is a programming language. In XML, both schema and the documents that conform to them exist as text files that can be edited by hand and by computer. In Java, classes are defined in text but the objects that are instances of the classes only exist in computer memory and so are not directly accessible by human hand. This means that some features, such as restrictions, that are directly attached to elements in XML schema but are actually just passive guidelines to what should be put in the element can be implemented in other places in a Java class but can actively control the values that can be put into a variable. Because even Java objects that are designed purely for the storage of data exist solely in memory, the only way to get data in and out of them are via methods. Methods and constructors, which are special methods for creating an object with some initial values, can be designed to put values into

variables in a class or give out the values of the class's variables. This means that the methods could also include code to check or even alter a value being put into a variable based on some conditions similar to restrictions in a schema. This ability of Java, being able to operate on values could allow X-Ninja to implement features inherent to schema that are not available at a basic level in Java.

The syntax for methods and constructors in Java is another feature that the translation tool will have to generate because they will contain certain context sensitive factors. Firstly, a method is declared as follows.

```
public int getValue()
{
        …
        …
}

public void setValue(int newValue)
{
        …
        …
}
```

Because of the reasons described above, any classes that the translation tool generates from XML Schema provide methods to get and set the values of variables held in them. For those methods the translation tool has to determine factors such, as the type of the value a 'get' method will return (i.e. the type of the variable the value comes from) or similarly the type of the parameter that a 'set' method takes in, and place an appropriate type name at the correct place in the code for the methods. It also must do the same thing as it does after writing a class, remember to close the braces around the method after writing the contents. However, the way this is done differs between classes and methods.

This concludes the overview of the structures that are available in Java. There are further features of Java that are termed structures such as arrays, which can hold multiple values of the same type in one variable, but the structures that are essential to the language have been covered. As has been shown there is less defined variety in Java but this is mainly due to Java being a programming language and so being far more powerful than XML which is purely a data description language and has no functionality. In fact, because XML documents are purely text document, specialised technologies have been developed to allow computer programs operate over XML effectively. The X-Ninja translation code uses the Document Object Model.

DOM

The Document Object Model is a way of breaking down an XML document into a representation useable by computer programs. The DOM takes a document and parses all the elements in it into an in-memory tree representation. Each element from the XML document is held in the tree as a node and any content the element had are children of that node.

DOM itself is an abstract, language non-specific standard supported by the World Wide Web Consortium (W3C). The X-Ninja translation tool uses a Java implementation of the DOM provided in the org.w3c.dom Java package that is provided with the Java 2 Standard Edition release. This parses an XML document into a Document object containing a tree of Node objects which can be operated using various methods to get and set values, child nodes, associated objects holding attributes etc. Further sub-classes of Node define Element, Attribute and Text objects (among others).

Project Details

X-Ninja's function

The purpose of the X-Ninja translation tool is to convert the types and elements in an XML Schema into classes and variables in Java code. Types will be mapped in the following way.

XML	Java
Element containing a complexType definition	*Class definition*
Element containing a simpleType definition	*Class definition*
Element containing a simple element definition	*Variable declaration*
Element containing a simple attribute definition	*Variable declaration*

For each element that describes complex or simple type, as well as a new class being generated for that complex or simple type, a variable is declared in the class corresponding to the elements parent that holds an instance of the new class. This is because, in XML, when a new type is described within an element it describes the fact that the parent contains an element of the new type. The variable translation of a simple element or attribute description is declared in the class corresponding to the complex element parent of the element describing the element or attribute. Because any element that is allowed to contain another element or an attribute is a complex element by the rules of XML Schema there is no danger of the parent element mapping to anything other than a class. However, because all XML Schemas have a root '<xs:schema>' element that all the rest of the content of the schema is contained within, simple elements and attribute descriptions can exist at the top level of the schema (i.e. not within a complex type description). To catch the variables translated from these top level elements and attributes, a class (termed the base class) which roughly corresponds to the schema's root element is generated. The base class holds both variables from these isolated simple elements and attributes, and instance variables of classes generated from complex types at the top level of the schema.

While the mappings of structures between the two languages are fixed in the current version of X-Ninja, the mappings of types is user definable via the use of custom mapping rules. By writing an XML document that conforms to the schema file 'rulesFormat.xsd', a user can control which Java type each XML base type maps to in the form <XML type>Java type</XML type>. The Java type can be any name, either a base Java type or a class name to map to an

object. Of course, if the value is set to an unknown name and a class of that name isn't present when the X-Ninja – generated classes are compiled there will be errors.

X-Ninja's mechanism

translator:

The main class of the X-Ninja translation tool is 'translator'. All of the other classes in the project (except for the example interface) are used in some way by translator during the execution of the program; the main flow of control is mostly governed by translator. While the main method of the translator class does allow some of the functionality of the program to be used from the command line, the program has been designed as a tool that can been plugged into larger Java programs. To access most of the tool's usefulness it should be used from within another class.

A new instance of the translator class should be constructed for each schema file that is to be translated. The translator class has several constructors to cover many different combinations of the three possible parameters a translator can be constructed with. It is required that a 'treeHolder' object holding a DOM tree of the schema to be translated is the first parameter to the translator object, otherwise the translator would have nothing to translate. It is also possible to pass a 'rules' object as a parameter to the constructor containing any custom mapping rules that are to be used during the translation. This is not necessary as, if no 'rules' object is set, a default one can be used or a custom one can be added later using the 'setRules' method. The third possible parameter is a string parameter that can be used to set the name of a base Java class that the will be generated by the translator. This parameter is also not compulsory, if not set the default string "base" is used and the name can also be set after the object has been constructed by calling the 'setBaseName' method.

Once a translator object has been constructed there are further options that can be set such as the output directory for the generated files, whether or not to generate constructors and access methods in the classes and whether to apply restrictions found in the XML to the Java code. There is a method to set each one of these options that can be called by whatever code is using the translation tool.

Once the translator object is configured to the user's preferences, the translation process is started by calling the translator's 'parse' method. This method begins the process of iterating through the nodes in the DOM tree and resolving their content to accurately work out a Java equivalent. The document's root node is acquired and then each of it child nodes is sent in turn to the 'parseNode' method. 'parseNode' is a method that is used several times during

the translation, most nodes in the tree will pass through it at some point. ‘parseNode’s function is to decide whether the node is an element and if so of what type it is. Once the type of element has been determined that are classes containing methods designed to extract information from each of a complex type description, a simple type description, a simple element description and an attribute description. There is also a class of methods to use if the node being parsed is an order indicator node and if the node is a ‘restriction’, ‘extension’, ‘complexContent’ or ‘simpleContent’ node its effect is ignored and its children are parsed. This ignoring does not mean that the translation tool does not take elements of these types into account but, as described later in the text, the effects of elements of these types will already have been evaluated when their parent elements were being examined.

As will be described later, each of the classes dealing with the four main element types will continue to pass nodes back through the ‘parseNode’ method until the end of all tree branches have been reached. This makes up the central loop of the program. Once this translation loop has gone through the entire document, another of the translator class’s methods is activated. Because of the way the tool generates and writes the Java translation of an element as soon as it has finished parsing that element, sometimes placeholders have to be put in the Java files instead of valid code. In XML Schema it is possible to describe an element by simply having a reference to an element fully described elsewhere, either in the same schema or an imported one. The translator keeps details of every element that it parses as it works through the DOM node, so if it comes across a reference it can look up the element that is reference to in it’s record. However, if the element reference to has not yet been parsed by the translator it cannot provide details of the element and so a placeholder string is written into the Java code where details need to be filled in. Then, once all schema information has been evaluated by the translator, the translator’s ‘resolveRefs’ method goes through a list of all the Java code files that have been generated, reopening each and checking for these placeholder strings. If any placeholder strings are found, the element referred to (which the placeholder string contains the name of) is looked up again. If the element is now found, the placeholder string is replaced with the appropriate Java code. If the element still isn’t found then an error has occurred or the reference was never valid in the first place.

That is the main process of a translation. The translator class has several other methods however, utility methods available to carry out common tasks required by other classes. Because the translator is a central point for the whole program most of these remaining methods are there to allow other classes to communicate with each other through the translator object (and will be mentioned during the explanation of the classes that use them).

treeHolder:

The purpose of the treeHolder class is too collate all the relevant information for an XML Schema or document before it is parsed. The treeHolder is constructed with a string giving the location of the schema file to be used, which it opens and builds a DOM tree from using the Java DOM implementation. The treeHolder then looks through the DOM tree for any parts of the schema that imports or includes other schema documents. If any are found, these schema files are also opened and parsed into DOM trees, which are then stored in an array within the treeHolder. After successfully loading all the schema information, the treeHolder object then acts as a wrapper class around the DOM tree for the central translator object.

rules:

The rules class holds the mapping rules between the XML and Java structures and types. When a rules object is constructed it can be passed the location of an XML document containing user defined mapping rules or it can be constructed without such a parameter to create a default set of rules. If the custom rules file is indicated, the rules object constructs a treeHolder object to parse the document for it. The default mapping are held in two hard-coded string arrays, one of the names of XML type and the other of their corresponding Java types, which are loaded in pairs into a hash map at the construction of the rules object. If there are custom rules, the DOM tree in the treeHolder is searched through for mapping definitions and if any are found, values in the Java types array are update accordingly before the hash map is populated. The rules object then has a 'getEquivalent' method that takes in the name of an XML type and returns its Java equivalent from the hash map. This method is what is used by the classes that generate Java code to look up which types new variables should be.

codeWriter:

The codeWriter class could be seen as an elaborate wrapper class around a Java.io.PrintWriter object. The purpose of the codeWriter class is to take care of all the file operations involved in creating and writing into '.java' files. A codeWriter object can be constructed with the parameter-less constructor, which simply sets up the codeWriter ready for used when it is needed. After construction, options can be set on the codeWriter object the same as those that are set on the translator object. In fact, the methods to set the options of output directory, whether or not to generate constructors and access methods and whether to apply restrictions are called directly from the corresponding methods in the translator object. This is an instance of the translator object being used as a go between, in this case between the codeWriter and whatever code is using the X-Ninja translation code.

A new file is created by the codeWriter when one of the classes actually generating Java code calls the 'makeClass' method. The method takes in two strings parameters, the first is

used by the codeWriter as the name of the new class and the second should be a string a Java code to declare the new class. The codeWriter creates a new file, using the name parameter with ".java" appended, in the directory specified by the output directory option. If the output directory has not been set the result will be that the new code files will all be created in the X-Ninja home directory. Once the new file has been successfully created and a PrintWriter object attached to it the string of declaration code is written into the file to open the new class. The codeWriter keeps track of how far code should be indented to keep layout of the Java code neat by virtue of a 'tabcount' variable. The 'tabcount 'is incremented once the class declaration code has been written to file and any code written to the file after that point will have than many tab characters prefixed to it before being written.

However because, in XML, new elements can be described within other element descriptions it is possible that the 'makeClass' method may be called while there is already a file being written. When this happens the currently open PrintWriter and some variables describing it (such as it's current tab count, the file name, and details of the variables so far declared in the file) are put into a fileHolder object and stored in an array with similarly open files. Then the new file is created and has a variable set to keep track of the index of its parent file (the file open when this one was created, the file just added to the array) in the fileHolder array.

Once a class file is open, further code can be written into it using the 'write' method. This is a very straightforward method that simply takes a string and writes it into the file. However, every time the 'write' method is used to write a variable declaration to the file, the 'addVariable' method should also be used to inform the codeWriter of details about the variable being added to the class. This becomes important when the codeWriter comes to write the class's constructor and access methods.

Once the code generating classes have finished declaring variables, the 'endBrace' method of the codeWriter is called. The method is so named because one of its functions is to decrement the tabcount variable and write the closing brace of the class to the file. But before it does that, the 'endBrace' function also calls the 'makeConstructor', 'makeSetMethods' and 'makeGetMethods' methods to generate and write code for the class's constructor and access methods. Once the open file has been completed, all methods written and closing brace appended, the function checks whether there was a parent file associated with the file that is being closed by looking at the parent index variable. If there is a value in the variable then the PrintWriter and other associated information in the fileHolder stored in the array of parent files at the index specified is loaded and made the active file again. If there is no parent index set this means that this is the base class being closed and the translation has been completed. In this case, the codeWriter now calls the 'resolveRefs' method in the translator.

The methods for generating the constructor and access methods do by iterating through a linked list of variableHolder objects stored in listNodes. The list is populated using the 'addVariable' method mentioned earlier to register with the codeWriter the variables that have been declared inside the class currently being written. The 'writeConstructors' method starts by writing a parameter-less constructor into the file and then starts work on a constructor that takes all values for all the variables in the class as constructors. Firstly it goes through the list of variables to obtain type and name information for each to build up a string that will be used as the parameter list on the constructor. Once this has been done the code to open the constructor, up to its opening brace (including the list of parameters), is written to the file. Next code is generated to populate each variable in the class with the corresponding value from the parameters; again this is done variable-by-variable – line-by-line. This part of the Java code being generated is also the place where any code modelling XML restrictions should be added. To do this, the 'applyRestrictions' method is used (described later) for each variable before the code is written to file. Once all the variables in the list have been evaluated and code for them put into the constructor the closing brace for the constructor is written to the file.

The 'writeSetMethods' method is in some ways similar to the 'writeConstructors' method. 'writeSetMethods' generates a 'set' method for each variable associated with the class, so each method only has one parameter, the new value for the variable that the method changes. Code implementing XML restrictions must also be used in the 'set' methods because they are used to change values in the variables and so 'writeSetMethods' uses the 'applyRestrictionsSetMethods' method to do this.

The 'writeGetMethods' method again generates a method for each variable associated with the class being written. These methods are designed to simply return the values of each variable and so the method must declare each method to have a return type the same as that of the variable it is returning the value of. The code within the methods is very simple to generate as it consists of nothing but the keyword 'return' followed by the variable name.

The 'applyRestrictions' and 'applyRestrictionsSetMethods' methods take in a string parameter and a variableHolder parameter. The string should be the current code being used to assign a value to the variable being processed; the variableHolder object should be that which holds information, including restrictions, about the variable. The 'applyRestrictionsSetMethods' method is really just a wrapper around the 'applyRestrictions' method that splits off some code from the 'set' method code leaving a string of code in the same form as those used in the constructor. This string can then be passed into the 'applyRestrictions' method and the extra 'set' method code can be re-concatenated onto the resulting string. The 'applyRestrictions' method checks the variableHolder object it is passed to see if there are any restrictions linked to that variable. If there aren't any, then the original code string passed

in is returned as the result, however if the linked list containing restrictionHolders in the variableHolder is not null the restrictions have to be parsed. The method steps through the linked list checking the type of each restriction and generating code for each. Some types of restriction cause code to be written that alters the value due to be stored in the variable before the variable is populated (e.g. a fractionDigits restriction). On the other hand some restrictions cause a conditional clause to be put around the population of the variable that only allows the value to be assigned to the variable if it satisfies certain tests (e.g. enumeration restrictions). Once all the restrictions have been evaluated and code generated for each has been added in the appropriate places of the original code string, the newly expanded code string is returned from the method.

javaSyntax:

The javaSyntax class is a class of static methods which holds strings of Java code for various situations for other code generating classes to fetch Java code from. The code strings are stored in a hash map, referenced by strings that describe where the code string should be used. For example, the code for declaring an integer variable at the start of a class is stored under the key “int” and the code for assigning a value to an integer variable in a constructor method is stored under the key “cint”. Before the javaSyntax library can be used the ‘initialise’ method must be called to populate the hash map using hard coded function calls that add the code strings. After it has been initialised, code can be brought out of the directory by calling the access methods ‘getSyntaxFor’, ‘getConstructorSyntaxFor’, ‘getSetMethodFor’ and ‘getGetMethodFor’. Each of these takes in the name of a variable type as a string and returns the appropriate string of code. If the variable type passed in does not correspond to any of the hash map keys, a generic code string is returned that allows type to be specified at a later point.

Because the code strings in the javaSyntax library have to be generic enough to be used for several different variables, identifier names cannot usually be specified in the hard coded strings. The exception to this is in ‘set’ methods when there is a parameter that only last for the scope of the method, all of which is hard coded in the string and so the parameter can be given a name. Where identifier and type names cannot be hard coded, placeholders are used in the strings in the places where the identifier names should be. For example, anywhere a variables name should be used in the code, the placeholder string “##name##” is there instead and in the generic, type-less strings mentioned above “##type##” holds the place where the type of the variable should be specified. In general the “##” placeholders will usually consist of double ‘#’ characters surrounding a word that corresponds to an attribute name in an XML element. This allows for convenient coding of the algorithm to replace these placeholders with the correct values. The method that does this job is usually the ‘insertVariableIDs’ method, one of the utility functions defined in the translator class for other

classes to access. 'insertVariableIDs' is called by one of the code generating classes and passed a string of code and a set of attributes (in the form of an org.w3c.dom.NamedNodeMap object). The method goes through all the attributes and looks for placeholder strings with names that correspond to the names of the attributes, when a match is found the placeholder is replaced with the value of the attribute.

elementOps:

The elementOps class is one of the code generating classes that X-Ninja uses. It contains the static method 'parseElement' which is responsible for generating the code for the translation of a simple element definition from the XML Schema. The first thing that the method has to do is check that the node it has been passed is that of an element definition. Because of the overlap between the ways of describing different elements in schema, the method checks any children nodes that the current node has to check if this is in fact a complex type description. If it does find that the node is the start of a complex type some changes are made to attributes in the node so that the translator class will recognise the node as a complex type and then the node is passed back through translator's 'parseNode' method.

If it is confirmed that the node is an element description, the method begins extracting the information it needs from the node to successfully translate the element into Java. If the element is fully defined in this node and any children it has then the method takes information on the type and name of the node and checks for any restrictions. If the element is a reference the method attempts to resolve the reference by calling the translator class's 'fetchPredefinedElement' method. Once the method has name, type and the other information it uses first the Rules class to get the Java type equivalent of the XML type and then uses the javaSyntax class to fetch the correct code for a variable of that type. Once the correct code is fetched and details about the variable filled in the code by translator's 'insertVariableIDs' method described above, the code is written to file by calling the translator's 'write' method, which in turn calls the codeWriter's 'write' method. If a reference cannot be resolved at this time, a placeholder string with the referred type's name inside it is put into the code string in place of type information. Once code has been written to file, the new details about the variable, such as name, type and any associated restrictions, are stored in a variableHolder object. This variableHolder is used to register the existence of this variable in the class being written by passing it, inside a list node, to the translator's 'addVariable' method, which passes it straight on to the codeWriter's 'addVariable'. This adds the variableHolder to the list of variables that the codeWriter uses when generating constructors and access methods for the class.

attributeOps:

The attributeOps class is one of the code generating classes that X-Ninja uses. It works in much the same way that elementOps does, containing a single, static method called 'parseAttribute'. The attribute is converted into a Java variable by fetching it's equivalent Java type and appropriate code and checking for restrictions in the same way that elementOps does. An attribute description cannot overlap with any complex type definition in the same way that a simple element description can so the 'parseAttribute' method doesn't have to check for this possibility.

simpleTypeOps:

The simpleTypeOps class is one of the code generating classes that X-Ninja uses. While both simple elements and attributes are of simple types, there is a further simple type case that this class is designed to deal with. In XML Schema, it is possible for a basic simple type to be defined that isn't an element or an attribute but actually just describes a type that elements and attributes can be of. This class deals with those type descriptions, translating them into Java classes that variables can hold instances of (i.e. can be of that type). The class contains a single static method 'parseSimpleType' to deal with the translation.

The resulting translated class has the name of the simple type it was translated from. Inside the class will be a single variable, of the Java type equivalent to the restriction base of the XML simple type, called 'value'. This variable will have the restrictions that distinguished the simple type from its base type attached to it and implemented in the class's constructor and 'set' method. The class is created by calling the translator's 'makeClass' method (which calls the codeWriter's 'makeClass' method directly), the variable is then written into the class through the translator and codeWriter's 'write' methods and then the class is completed using the 'endBrace' methods.

complexTypeOps:

The complexTypeOps class is one of the code generating classes that X-Ninja uses. The class contains a single static method, 'parseComplexType', which opens a new class in the same way that the simpleTypeOps does. However, the variables in a class translated from a complex type are made up of the translations of the elements and/or attributes that make up the content of the complex type. To facilitate this, once the new class file has been created and the class declaration written into it, the child nodes of the complex type are iterated through and passed in turn back through the translator object's 'parseNode' method. In this way the contents of the complex type are translated as each would be, as described earlier in the text, and the Java code that they are translated into is written into this complex type's open class file. Once all the children have been translated, control comes back into the

‘parseComplexType’ method that calls the ‘endBrace’ methods to complete and close the class.

classOps:

The classOps class contains some static methods that can be used by the code generating classes to determine certain properties of elements they are working on. classOps is so named because its original purpose was to extract the details of an extension in a schema and process that data to allow a Java class being created to be an extension class. However, the name is not quite so accurate now as classOps also contains static methods for extracting restrictions to be associated with variable.

The ‘checkExtension’ method takes in a node and the Java code that one of the code generating classes has so far produced for that node. Its job is to check the child nodes of that node it is given to look for any extension descriptions. The method checks if any of the elements in the first level of child nodes are ‘extension’ types. If none are found the method also looks for an nodes that are ‘complexContent’ or ‘simpleContent’ descriptions, this is because complex types in schema often prefix an extension description with one of these elements. If one of the ‘…Content’ elements are found the method looks through the next level of child nodes, again looking for ‘extension’ elements. If no extension if found, the original string of code is returned unaltered. However, if an extension is found new class declaration code is acquired, from the javaSyntax library class, to declare a class that is extended from another class. The classOps class also has its own variation on the translator class’s ‘insertVariableIDs’ method, called ‘insertInExtendedClass’. ‘insertInExtendedClass’ does the same job as the generic ‘insertVariableIDs’ but is specialised for extended class declaration code.

The classOps class can be used to check an element or attribute for restriction descriptions by calling the ‘checkRestriction’ method. The method functions in a similar way to that checking for extensions, it searches the child nodes of the node it is passed looking for a ‘restriction’ one. If a restriction is found, the ‘parseRestrictions’ method is then used to search *its* child nodes to collect details of each restriction type specified and the details for each restriction specified are stored in a restrictionHolder object. The method does not do any operations with code, none is passed in and the method doesn’t return any. Instead, a linked list of listNode objects is returned, the first holding a string denoting the base type of the restriction found and the rest each holding a restrictionHolder object with details of a restriction to be implemented. If no restrictions are found a single listNode object is returned which holds the string “no restrictions”. This list is returned from ‘parseRestrictions’ to ‘checkRestriction’, which then returns the exact same list to whoever called it.

orderIndicatorOps:

The orderIndicatorOps class was intended to implement the effects of the XML Schema order indicator elements - ‘all’, ‘choice’ and ‘sequence’. However this feature is not implemented in the current version of X-Ninja. While the class’s static method ‘parseOrderIndicator’ is called by translator’s ‘parseNode’ whenever it comes across an order indicator element, the ‘parseOrderIndicator’s only function is to ignore the order indicator element and pass its child elements back to ‘parseNode’ one by one.

fileHolder, variableHolder, restrictionHolder:

The various ‘…Holder’ classes are simple storage classes that are used as convenient receptacles in which to keep groups of related data (about files, variables or restrictions) while it is being stored or passed between different methods.

listNode, myQueue:

The listNode class implements a generic list node for a singularly linked list. The node can hold anything of the type Object in its head field and has a tail field that holds the pointer to the next node in the linked list. listNode linked lists are used in places throughout the X-Ninja code to hold lists of variableHolder, restrictionHolders and mixed types while they are being stored or passed between methods.

The myQueue class implements a simple, two-stack based queue. The queue is used by the program to hold details of files being written for when they need to be reopened for references to be resolved. A myQueue Object allows anything of class Object to be added to the back of its queue and allows for the Object at the front of the queue to be ‘peeked’ at or removed. The inner workings of the queue are implemented in the form of an ‘in’ stack and an ‘out’ stack. When an object is added to the queue it is pushed onto the top of the ‘in’ stack, when an object requested from the front of the queue it is popped off the top of the ‘out’ stack. If the ‘out’ stack is empty when this happens, the objects in the ‘in’ stack are popped off ‘in’ and pushed onto ‘out’ one by one. This means that the object from the bottom of the ‘in’ stack, the first thing added to that stack, ends up at the to of the ‘out’ stack and so is the first this to be removed, as is correct in a queue.

exampleInterface:

The exampleInterface class is not part of the main body of the X-Ninja translation tool. However this class is an example of how the translation tool can be used by other code. The exampleInterface class builds a small GUI application that uses the translation tool as its back end. The GUI allows a user to specify an XML Schema file that they would like to be

translated, an optional XML document that contains custom mapping rules, an output directory for the generated files and a name for the base class. The user also has the opportunity to set the options for whether or not constructors and access methods are generated and whether restrictions should be applied in the Java code.

This GUI was simply intended as a demonstration of a possible use of the X-Ninja tool but also proved to make testing the translation tool much easier because of its increased functionality over the translator class's limited command line interface.

Evaluation and Critical Appraisal

The original objectives of the project, as stated in the Project Description and Objectives document (see appendices) where as follows.

- To generate correct Java code defining classes based on the contents of XML Schema documents. The Java classes should have constructors and access methods.
- To allow the rules governing such a mapping to be defined in a file external from the program code, thereby allowing a user to customise the mappings to their own needs.
- To allow these rules to be edited by the user via 'the' user interface of the program.
- To extend the program so that it could evolve existing Java classes based on a parsed XML Schema.

However, during the planning stage of the project, as the concept of the translation tool became clearer, some of these objectives became less prioritised. Most notably, the decision was taken that the core of the project would be to work on the translation tool as a group of classes rather than an application. This meant that the project became to produce code that could provide its translation functionality to other programs, the production of a GUI front end for the tool was optional and would only be done as an example of how the core code could interact with other code. Also, a second decision was made that the format of the external rules file would be XML. Using a common format such as XML means that there are already several programs readily available that could provide a user-friendly interface for editing the file and so the emphasis on the project to provide such a thing was reduced.

At a basic level, the current version of the translation tool does successfully implement some of these objectives. The tool can successfully generate correct Java class code that can be compiled and put into use as data storage classes as intended. Custom mapping rules, governing the mapping between XML and Java data types, can be defined in an external XML document and imported into the program and have the desired effect. The project didn't reach an advanced enough state for the objective of being able to evolve existing Java code to be implemented in the allotted time.

However, when compared to other tools that have the same purpose as this translation tool, such as the tools implementing the Jax-B standard, X-Ninja has some serious shortcomings. Jax-B provides a complete interface to map between Java and XML data definitions. Originally based around the conversion of DTDs into Java code, Jax-B was recently extended to provide the same functionality for XML Schema files. Jax-B can successfully map every

possible feature of an XML Schema into some Java alternative and allows for extensive user control over the mapping. On the other hand, the X-Ninja translation tool does not have full support for schema features. Most significantly, this project currently has no support for XML namespaces, a technique that allows for elements with the same identifier name to be kept distinct when used along side each other as long as they where defined in separate schema. Currently, if this project was presented with two identical identifier names, even from separate namespaces, the advent of the second occurrence would overwrite the details of the first. This would most probably lead to compile time or run time errors when a user came to use the X-Ninja – generated classes.

There are also certain features of the XML Schema syntax whose effects are not successfully translated to Java code by the translation tool. Features such as occurrence indicators and order indicators whose effects are currently ignored by the translation tool. Theoretically, if an element is defined in the schema with the 'maxOccurs' attribute set to a value greater than one, the corresponding Java variable should also be able to hold a number of values greater than one, say as an array or list. At the moment however all elements, whether they have a 'maxOccurs' or 'minOccurs' attribute set or not, always map to a single type variable. Other available and significant attributes in schema, such as the 'use' (denoting whether an attributes use is optional or required in the element it is applied to) attribute in an attribute type definition, which is not currently picked up by X-Ninja.

Further, some of the features that the X-Ninja translation tool does implement are not done so very effectively. For example, a schema element that is described by an extension of an existing type is successfully translated to a Java class that is declared to extend the class corresponding to the element that was the extension base. However, the code within the extending class does not implement the extension in any functional way. The constructor of the extending class only invokes the constructor of the extended class by inherently invoking a parameter-less constructor. No data is stored in the variables gained by extension from the original class and, since classes generated by X-Ninja are for the express purpose of data storage, this means the extension has little practical effect.

On the plus side, the code generated by the translation tool seems to be very robust and hard to break. If the generated classes are intended for data storage then they should serve that purpose adequately. Also the translation of XML restrictions into the generated Java classes is carried out elegantly and very successfully considering some of the problems it could have caused.

Conclusion

In conclusion the project in its current state is not a significant achievement. Whilst it carries out its basic functions reliably and consistently well, there are serious shortcomings with both functionality and design. If there was more time available to work on the project, then many of the problems that have been highlighted previously in the report could be solved. However, with any significant length of time available the best approach would be a complete redesign of the approach to the problem. Many early design decisions taken during the project have turned out to have adverse effects on the achieveability of later features. For example, the mechanic of the program to generate Java code and write it to file line by line rather than to store generated code in-memory somehow (one possible alternative) meant that resolving references in the schema had to be implemented in a very inefficient manner.

In all, most of the problems and deficiencies in the project can be blamed on lack of design and lack of planning. I struggled to grasp a full understanding of the workings of XML Schema early in the project, which meant that any in-depth design was very difficult. By the time I began to feel comfortable with schema, the code process was well underway with no time left to start again. These are the major reasons why so many parts of the code seem to work around other parts. As I tried to add more features I often found that the way earlier features had been implemented sometimes specifically prevented me from taking, and often made it difficult to take the obvious approach towards the current problem.

Appendices

Appendix 1

PROJECT DESCRIPTION AND OBJECTIVES

The aim of this project is to create a tool that will convert types in the XML Schema to classes and variables in Java. The general view that will be started with is that each complex type in the XML Schema will be mapped to a class in Java, and simple types in XML will become variables in Java. There are several possible iterations of the project that may be attempted, each more complex than the last.

The first and most basic aim of the project would be a tool that takes in an XML Schema document, an XSD file, and creates a corresponding set of Java classes based on a fixed set of rules governing the mapping. The classes should have the standard functionality of 'set' and 'get' methods to access their data as well as constructors to create instances of the class. Because XML indicators (a feature of XML Schema that can limit the range of possible data that can be stored in an element) have no direct equivalent connected to Java variables, the restrictions they place on data will have to be implemented in the constructors and 'set' methods of the created classes.

The second aim of the project will be to allow the functionality described above but to have the rules governing the mapping be defined in a separate file, in whatever format is deemed suitable, so that different sets of rules can be used for different Schemas. This would allow a user to tailor the resulting Java classes to their requirements, perhaps so that they can fit alongside already existing classes or programs.

The third aim of the project will be to allow the rules governing the mapping be altered by the user via the user interface of the program. The tool will be written in such a way that the code providing the functionality of the program will be kept separate from the user interface, thus allowing the tool to be more easily integrated as part of a larger application or to allow different user interfaces to be dropped on top of the tool.

The fourth aim of the project will be to implement a mechanism in the code to cope with evolution of the mapping. This means that the program will be able to take an XML Schema and a set of Java classes that it has already generated from that schema and then check the two to see if anything has changed in either. If the program finds anything different in either the schema or the classes, new variables for example, it should be able to alter one or the other (which is altered could be a decision made by the user) so that they once again map to each other correctly.

PROJECT PLAN

Context Survey

Background Topics

XML:
eXtensible Markup Language is very much the craze of current Computer Science. Despite, or perhaps because of, it's humble beginnings and innate simplicity, XML is being used as a basis for re-implementations of anything and everything from database engines to operating systems and all things in between.
XML is known to many people as a close relation of HTML and though the two are connected the similarities these days aren't that numerous. XML is actually a name given to a group of interconnected technologies and specifications. At the core of this group is the XML Information Set (Infoset) which gives a syntax free definition of what a well formed XML document should be, defining terms such as an XML element, information items etc. and there roles in XML. Also in the group are various sub-languages, XML syntax for formatting data, with type systems and data structures, eXtensible Stylesheet Language for controlling how the data is represented on screen and various definitions such as the XML Linking Language (XLink), the XML Pointer Language (XPointer), XML Inclusions (XInclude) and XML Base (XBase).
A wider view of the XML family might also include DOM and SAX, both of which are discussed below.

DOM:
The World Wide Web Consortium (W3C) describe the Document Object Model (DOM) as "a platform, and language, neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents". In relation to XML, DOM is a set of abstract interfaces that model the XML Infoset and provide methods that allow for the creation and parsing of XML documents through programming. The DOM represents the XML Infoset as a tree structure held in memory, allowing in-memory traversal and modification of documents. Each node in a DOM tree represents an information item in the XML document, there are several different DOM nodes relating to different items in XML such as elements, attributes, comments etc.

SAX:
The Simple API for XML (SAX) is another set of abstract interfaces that are used to describe the XML Infoset via a set of methods. SAX uses a completely different approach to that of DOM. Where DOM builds a complete in memory structure of the XML document it is dealing with SAX is used to traverse the document, stopping at each piece of markup (angle bracket)

it finds and returning data to the application as it is found. This 'stream of data' approach means that memory is not used up in storing the XML data and that reading of the XML document is very fast. However this also means that SAX cannot be easily used to manipulate the XML data.

Similar Tools

The following are software tools already available, or in development, that perform similar tasks to those that this project is intended to implement.

JaxB

The Java API for XML Binding (JaxB) is a set of APIs and related tools for mapping between XML and Java. JaxB was originally implemented to deal with DTDs (Document Type Declarations) but, as the XML specifications moved away from dealing with DTDs and towards XML Schema, support for Schema was introduced and now DTD is no longer supported. JaxB works in a very similar manner to that in which this project is intended to work, taking in an XML Schema document and a set of binding rules as input and compiling "JaxB content classes in the Java programming language". The best known implementation of the JaxB API is the one provided by Sun downloadable from the java.sun.com web site.

The current version of Suns JaxB tools is the Beta 1.0 version. Perhaps because it is still only a beta version and it is being implemented by the same company as defined the API, development of the Sun tools has focused very much on implementing the full functionality of JaxB and the tools are quite reliable and stable. However there are no user friendly GUIs supplied with the tools and the only way to use the tools is via a command line interface. This means that a user would have to read and understand the 190 page plus associated documentation to be able to access the full functionality of the tools.

XML-SERIALSER 1.0

The XML-SERIALSER 1.0 by Adaptinet is a data-binding tool intended as an alternative to DOM and SAX for developers. The tool can be used to take data out of XML files and store it in Java classes for data storage and is designed to be a resource for other programs to use rather than a stand alone application itself. The SERIALSER first builds a class definition by taking either an XML Schema or a DTD as input into a stand-alone compiler, or via a plug-in for the Java development environment JBuilder. Once this class definition has been built, the XML-SERIALSER can be used to parse data from XML files conforming to the initial schema or DTD into instances of the defined classes as part of a run time system.

Because the tool is intended as an alternative to SAX or DOM it is designed primarily for use by other programs. This means that there is little or no UI for a user to use the tool on its

own. However, the tool does not have quite as much potential as SAX or DOM because SAX and DOM are APIs that can be implemented in many programming languages whereas the XML-SERIALSER is firmly routed in the Java language. Also, because the classes built by the XML-SERIALSER are primarily intended for data storage and retrieval by the program that created them, the tool does not automatically generate constructors and access methods that take into account restrictions on values.

XML Visual Basic Class Generator 1.0

The XMLVBCG by Genialt is a tool that will automatically generate a skeleton Visual Basic Class to hold data from an XML file. The program differs from the intended purpose of this project in that it generates a new class from scratch for every XML data file rather than building a class definition from a schema and the constructing instances for each XML file. The tool is designed as a stand alone application and so has a basic user interface, however options on the mapping are set by passing the tools Visual Basic code and so the user would have to be familiar with the language to make use of the tool.

As the XMLVBCG's creator puts it "The program is developed as a dirty, little tool to quickly create the skeleton for XML class design". Perhaps as a result of this, the tool has several bugs and problems that are unlikely to be addressed in any program updates as development of the program is currently on hold pending user feedback.

Problem Specification

Conceptual Model

The task of taking an XML schema document and turning it into a compilable Java class can be broken down into several stages. First of all, XML Schema has the possibility that not all the elements being used in the schema document are defined in that same document, they can be referenced from other schema documents whose locations on the internet are declared at the start of the XSD. This means that if only one XSD is given to the tool as input, it may have to fetch the other required XSDs via FTP or some other similar mechanism. Once all the required schema documents are available the next step would be to parse the XML in them into a format usable in the rest of the program. As there are already two available APIs for parsing XML data, SAX and DOM, this is not so much of a problem that this project will have to struggle with. After the XML data has been parsed, using either SAX or DOM, the next step would be to translate the XML into its Java equivalent. This translation would be done according to the rules set for the mapping. If the tool is to be fully customisable then the rule set will have to describe how every supported XML information item is to be mapped into Java. The exact format of these rules will be described later in the document.

Functional Requirements

Input:
The completed tool will accept several different files as input. The most obvious input is the XML Schema that the user wants converted, this will be passed to the tool in an XSD file. Because some XML Schema definitions reference elements defined in other XSD files the tool will be capable of finding such references and fetching other XSD files for input via FTP.

The tool will also take in a file defining the rules governing the conversion between XML Schema and Java. This file will be an XML file that should comply with the following XML Schema definition.

```
<Schema xmlns='http://www.w3.org/1999/XMLSchema'
        targetNamespace='cjm:shproject:rulesdefinition'>

        <simpleType name='yesno' base='NMTOKEN'>
                <enumeration value='yes' />
                <enumeration value='no' />
        </simpleType>

        <complexType name='simpleTypeMap'>
                <element name='javaType' type='string' />
                <element name='mapToVariable' type='yesno' />
        </complexType>

        <element name='typeMap'>
                <complexType mixed='true'>
                        <element name='attibute' type='string' minOccurs='0'
maxOccurs='unbounded'/>
                        <element name='children' type='simpleTypeMap' minOccurs='0'
maxOccurs='unbounded'/>
                </complexType>
        </element>
</schema>
```

Lastly the tool will take in existing Java classes as input in instances when the user wants the tool to adjust existing classes rather than create new ones.

Conversion:
The tool will parse through the XSD files it is given as input using the DOM discussed above. It will then write Java code into a file/buffer created by looking at the types, elements and

attributes in the XSD files and generating the corresponding Java. There will be a library of generic Java code which the tool can draw on, code for declaring a class definition, code for declaring a variable etc.; it will decide which piece of its generic code to use by looking up the name of each information item found in the input file in the rules file. The rules file should hold information about what Java element that XML item should be mapped to and the tool will then know how to customise it's generic code for that information item (name, values etc.). If the input XSD file contains an item that the rules definition does not cover then the general mapping rule defined in the rules will be used. If there is no general case defined in the rules file then a super-general case, which will be hard-coded into the tool, will be used.

During this process, standard 'set' and 'get' methods will be generated for any variables that are created and a constructor for each class will be generated. Depending on options set in the rules file, XML indicators may be used to set up restrictions on what values can be put into the created variables, this restrictions would be implemented in the 'set' methods and in the constructor.

Lastly the generated Java code will be complied as a final check that it is correct. The project is intended to always generate correct Java code, if the code does not compile this would be a bug in the program rather than anything that can be checked for and corrected dynamically as the tool is running.

Output:
The tool will output the generated Java code and compiled classes into a directory specified by the user.

Non-Functional Requirements

Usage:
The main emphasis of the project will be to produce a tool rather than an application. The tool will be a set of classes that offer methods that will allow other classes and applications to access the functionality provided by the tool. By coding the functionality in this way it will allow the tool to act as a plug-in to other programs and also allow us to build more than one GUI for the tool, perhaps different ones tailored to different environments, which can easily sit on top of the main functionality. What this means is that usage of the finished tool will be in two distinct ways, either by calling the function classes or by running an executable to provide a GUI.

Any GUI that are supplied with the finished tool will allow the user to specify the initial XSD file and the directory into which the output Java files should be put. As the functionality of the project increases further options, such as custom rules files, already existing Java files to be

evolved etc., will also be specified by the user through the GUI. A further improvement to the GUI would be to implement a rules editor, giving a simple GUI to alter the rules instead of having to hand code them into a file.

Hardware:
The finished tool should not need any specialised hardware to run. Because of the distributed nature of some XML Schema Definitions the tool may need access to an internet connection through which it can download further XSD files that may be referenced in the file it is initially given as input.

Supported Platforms/Development Platforms:
The project will be developed in Java, using the Java Swing libraries for any GUI programming. Because Java is platform independent the tool should be capable of running under any operating system that supports Java.

Documentation:
A full set of development documents outlining the development process and any difficulties encountered during it will be provided with the completed software. Also, a comprehensive user manual detailing basic operation, editing/creation of custom mapping rules and any further advanced functionality that is implemented will be produced.

Error Tolerance:
In the case of errors such as corrupt or incorrect input, the user will be given a message detailing which input file the tool believes to be the problem and asked to check or change their input. In normal running the tool should be tolerant of users passing input or attempting to interact with the program at inopportune times (e.g. User clicking on buttons in a GUI while the program is in the middle of it's conversion process). If exceptions are thrown by non-GUI code during the conversion process then the process should normally be stopped, as any error in this process would produce incorrect or incomplete Java code in output, which would be useless.

Modular Design

I/O & Parsing Module:
This module will be responsible for dealing with input files. It will have two distinct functions: parsing XML documents and parsing Java class files. In parsing XML documents the module will be responsible for dealing with both the XSD files that the user passes in to be converted and also for dealing with any rule definition files, in XML format, that the user might specify to be used. It will do this using the DOM API, probably by employing an already available Java implementation of the DOM rather than requiring a new implementation to be written. The

module will be called by other modules to get the information from these XML files and so will provide several methods to provide pieces of data in different formats. In parsing Java classes the module will be required to de-compile the class files into a format which can be edited, most probably using an already available third party Java de-compiler such as the one provided by Microsoft. This will allow the module to produce text format Java syntax files which can then be edited in the same way that the tool will write to new such classes. Again methods will be implemented to allow other modules to fetch data from the Java files and also to allow other modules to write into these files. In cases where there are no Java files passed as input, this module will be responsible for creating and writing to new Java files.

Conversion Module:

This module will be responsible for the actual computation of deciding how to convert XML Schema into Java code. The module will contain a library of generic Java code, most probably as String objects, that will be used to build the create Java files from. The conversion module will use the I/O & parsing module to get information out of the XML file defining the conversion rules and will store that information in 'rule' objects, along with the super-general rule object that will be hard coded into the module. It will also use the functionality of the I/O & parsing module to take information items from the XSD input file(s) and build the required Java code by pulling suitable strings of Java from its library, modifying them for the specific case and then passing the string to the I/O & Parsing module to be written to the Java file.

For example, if there is a complexType with the name 'example' defined in the XML code, the conversion module will look through its rule objects to see if there are any governing the 'example' type. If there are, the rule may specify that this 'example' complexType should be converted to a class in Java and so would get the class definition string from its library. This string may be of a form similar to "public class ::name:: {" and so the module would then replace the '::name::' part of the string with the name of the complexType, 'example' and then would pass the string "public class example {" to the I/O & parsing module to be written to file. At this point, because a class definition has been started, the conversion module knows that anything it finds within the complexType 'example' in the XML will be converted to an object or variable within the Java class 'example'. Every time an XML information item is mapped to a Java variable the conversion module would automatically generate 'set' and 'get' methods by the method of fetching generic code strings and adding name and type information. Once all the items inside the complexType 'example' had been parsed the conversion module would add a constructor and close the class definition.

Wrapper module:

The wrapper module will be an interface module; it will provide the access methods that would be called by an outside user who wanted to use the tool or by a UI that may be slotted on top of the tool to make it a stand alone program. The module will basically consist of

various ‘start’ methods that take in the different kinds of input that the tool can accept and possibly provide output as a result of the conversion. For example, the most basic method might take just an XSD file as input, which would then be converted using a default rule set and the generated Java classes saved to a default folder. A more in depth method would take multiple XSD files, an XML rules definition file, some Java class files and a string denoting where the user would like the resulting output saved to. It would perform the conversion of the XSD files based on the rules in the XML file, update the Java class files, save them to the indicated directory and then return details of what changes where made to the class files as output.

GUI:

As part of the project, there will also be an example GUI to show how UIs will be able to be placed on top of the base functionality of the tool. The fully realised GUI will allow the user to specify their input files via standard file browsing objects, to define their own rules for conversion in a rule editor and to give details of exactly what items in the XSD input became what in the Java output. The idea of the rules editor would be to allow the options for several items in the XSD files to be set by selecting from drop down lists rather than writing XML in long hand.

Software Engineering Plan

Evolution and Testing

The project will be in a constant testing process, as each module of code is developed it will be tested to make sure that it successfully provides the functionality it is designed to. As soon as a functioning system is in place, the project will be developed in an evolutionary manner, whereby the first iteration will be given actual input to deal with, containing any and all types of input it will be expected to deal with in general use, and expected to perform in the designed manner. When problems or failures are found the code will be improved to solve them until that iteration of the tool works in the expected manner. Once this is done, the code will be evolved to support further functionality and the testing/debugging process will start again.

Version Control

The project will be developed under the CVS system. CVS has an advantage over manual versioning in that it logs every change to every piece of code separately. Firstly this takes the burden off the developer to decide when to declare that the project has reached a new version, as they would have to do if versioning manually and also, when a fatal error is made in the code, CVS allows the rolling back of single files to revive the project as opposed to rolling back to an earlier version of the entire project as is common in manual versioning.
In order to defend against severe device failure, each time the system reaches a new level

functionality, backups of the entire code base will be made to CD-R. In this way, even if the machine holding the CVS repository was to fail, as working version of the project would still be available.

Fallback Plans

The objectives of the project are segmented in such a way that they should be achieved one at a time after distinct development periods. This means that, even if there was a serious problem that impinged on the development time of the project, such as severe illness, some of the original objectives could still be achieved and a working tool would be produced.

Hand-in Deadlines

Semester1

11th October 2002: Project Description and Objectives

30th October 2002: Project Specification and Plan

4th December 2002: Interim Report No.1

Semester2

12th March 2003: Interim Report No.2

23rd April 2003: Project Report, software and documentation

12th May 2003: Project Presentation

Project Milestones.

31st October - 17th November 2002: Detailed planning of module functionality and interfaces

18th November - 1st December 2002: Coding of I/O & parsing module first iteration (DOM implementation and file I/O)

2nd December - 13th December 2002: Begin coding of Conversion module first iteration (Using hard coded rules and not implementing XML indicators in Java constructors and access methods)

14th December 2002 - 9th February 2003: Christmas holidays, revision and exams

10th February - 23rd February 2003: Complete coding of Conversion module first iteration

24th February - 9th March 2003: Coding of I/O & parsing module second iteration (Multiple XSD file input and parsing of user defined rules files)

10th March - 23rd March 2003: Coding of Conversion module second iteration (Fetching XSDs referenced from other XSDs, converting based on user defined rules and using XML indicators to build constraints into Java constructors and access methods)

17th March - onwards: Writing project report and documentation

24th March - 30th March 2003: Coding of I/O & Parsing Module third iteration (Parsing Java class files given as input)

31st March - 13th April 2003: Coding of Conversion Module third iteration (Interpreting and evolving existing Java classes)

14th April - onwards: Final testing process

INTERIM REPORT 1
4/12/02

At this point in my project I have begun experimenting with the base technologies, such as XML Schema and DOM on which a lot of my project will be based. I am currently making an in depth exploration of the Document Object Model and the associated APIs and classes implemented by sun, and have written a few small programs to test my understanding of the concepts behind the DOM.
I am now confident that my understanding of the project is much better, as at the beginning of my plan I didn't have a very good grasp of the actual technicalities of XML Schema and the methods for parsing XML, such as SAX and DOM. After spending the last few weeks researching these issues, I now feel more capable of making a start on the actual coding of the project, and hopefully will be able to develop the experimental programs I have written towards the first coding objective of having a preliminary mapping working.

In relation to my project plan and time table, I'm currently behind schedule in that the time I have spent researching I was due to spend working towards this first objective. However, I think this is mostly a fault of bad planning as I allocated very little time to initial reading but gave myself a month to write code which I now think will take significantly less time than that. Further more, I'm now planning on spending time in January working on the project that I didn't originally schedule and so I should be much closer to the deadlines of my plan by the start of next semester.

INTERIM REPORT 2
12/03/03

At this point my project is heavily behind schedule. The project is currently stalled at the point of deciding conversion rules for the mapping from XML schema into Java source code. The process is proving to be more complicated than I first anticipated due to tags being affected by different namespace prefixes and other matters of syntax.

According to my original project schedule, the project should currently be on its second iteration, which would be improving the basic working program to take account of loading in further XML schema referenced or imported into the base schema file. However, this particular feature has already been implemented in the version of the project that I am working on as it proved to be essential to the loading of the schema files and construction of the data structure to hold the DOM trees which was the first task I worked on.

Considering this rearranging of tasks in the schedule, the project is not as heavily behind schedule as the Project plan would suggest but is still running late by approximately 3 weeks.

Appendix 2

TESTING SUMMARY

Throughout the development of the project, testing has been carried out via the use of example input and running of the code. All of the non-static classes used in the project where given main methods which where designed to create and experimental instance of the class and simulate or actually carry out the classes task. For example, the treeHolder class is designed to take in a document in XML format and parse it into a DOM tree. To test this class the main method constructs a treeHolder object with an example XML schema file to check that no errors occur during the class's constructor or parsing method (which is called by the constructor). While the classes where still being coded, several print statements where put into the code that would print to screen the contents of important variables at certain points throughout execution. In this way I could manually check that the code was doing exactly what I wanted/expected it to do and in those times where it didn't, I was able to see the point at which values deviated and so would know which section of code to look at.

Once the project got nearer to completion and static classes became involved, this isolated checking of classes had to be revised. Instead I began running the program in its entirety to see the results as I added more and more functionality. I continued to use print statements in any new sections of code that was added to check their behaviour and, if everything behaved as I expected, I could then look at the actual output of the translation tool. If the program had run successfully but the Java code output contained errors then the design of the responsible parts of the program had to be rethought and retried. This process made up the main body of the develop-test-develop iterative cycle.

Finally in the dedicated testing and bug fixing period at the end of the project, I defined a series of example XML Schema files with which to test the program. The example schema contained various different features, and different combinations of these features, of XML Schema to see how the tool would react to having to take several possible routes through its code. If an example schema caused problems, I would break down the combinations in the example file and retest until it could be determined exactly which feature or combination had caused the problem. The bug would then be fixed.

If the program performed correctly with a test file, for all combinations of option settings, it would be run again with the same file to ensure consistency and then the produced Java code would be compiled to check its complete validity.

Appendix 3

STATUS REPORT

The X-Ninja conversion tool is currently incomplete. The following features or objectives have yet to be implemented fully, if at all.

- Extension types do not fully extend their parent class in that they have no ability to use variables inherited from the parent.
- XML order indicators are parsed but their effects are not implemented in the generated Java code.
- XML occurrence indicators are not parsed.
- XML namespaces are not supported.
- A full and comprehensive user interface (allowing user friendly editing of mapping rules) has not been implemented.
- The ability to operate over existing Java code and add to/alter it has not been implemented.

Appendix 4

MAINTENANCE DOCUMENT

Type of error	Action to be taken
Generated Java code has compile-time/run-time errors.	Determine which part of the Java code is incorrect and refer to the relevant solution in the table. For details of how all code should function refer to online JavaDoc and the comprehensive comments within source files.
The Java code contains sequence(s) of characters of the form "##name##"	The error could be with the translator.insertVariableIDs() method or one that calls it. Find the XML element that is being translated to this incorrect line. If it is an element the error could be in elementOps.parseElement(). If it is an attribute the error could be in attributeOps.parseAttribute(). If it is a simple type description the error could be in simpleTypeOps.parseSimpleType(). If it is a complex type description the error could be in complexTypeOps.parseComplexType().
The Java class declaration uses incorrect syntax / contains the wrong class name	Find the XML type description that is being translated. If it is a simple type description the error could be in simpleTypeOps.parseSimpleType() or codeWriter.makeClass(). If it is a complex type the error could be in complexTypeOps.parseComplexType() or codeWriter.makeClass().
The Java class is left open at the end of the file	The error could be in the codeWriter.endBrace() method or one which calls it. If the class is based on a simple type description, the error could be in simpleTypeOps.parseSimpleType(). If the class is based on a complex type the error could be in complexTypeOps.parseComplexType().
A variable declaration is of the wrong type	If custom mapping rules are being used, the error could be in rules.parseRules(). If

	default rules are being used, check the arrays types[] and maps[] in rules for correct values in corresponding indices.
A variable declaration is of the type 'Object'	The type being looked up in the rules class is not being found. Check the type mappings in any custom rules being used or the default mappings in the arrays types[] and maps[] in rules. If these are all correct, the method calling rules is dealing incorrectly with the value it is being returned. The error could be in elementOps.parseElement() or attributeOps.parseAttribute().
The use of a variable in a constructor / access method is of the wrong type / name	The variable has been registered with the codeWriter using incorrect details. Check the addVariable() call in either elementOps.parseElement(), attributeOps.parseAttribute(), simpleTypeOps.parseComplexType() or complexTypeOps.parseComplexType().
The syntax for a constructor / access method is incorrect	Check the syntax stored under the appropriate key in javaSyntax. Check codeWriter.writeConstructor() / codeWriter.writeSetMethods() / codeWriter.writeGetMethods().

BASIC CRITERIA

Understanding of the Problem	c
Translation of Schema to classes - nothing about giving a typed view	
Proper Software Engineering Process (including Plan)	c
OK but not startling	
Achievement of main objectives	C
done - but very unambitious	
Structure and Completeness of the Report	B
OK	
Structure and Completeness of Presentation	B
Yes structure is good.	
ADDITIONAL CRITERIA	
Knowledge of the literature	C
Literature surver covers main areas but is skimpy	
Critical evaluation of previous work	C
OK	
Critical evaluation of own work	c
ok	
Justification of design decisions	c
Solution of any conceptual difficulties	C
Achievement in full of all objectives1	C
Quality of Software	C
Ambition and Scope of Project	C

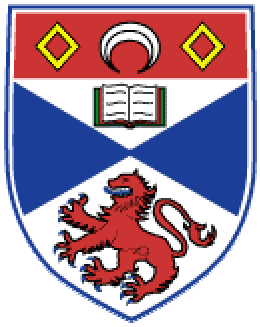
Grade = 11.

Text to Video Instant Messaging System

Emma Russell

23rd April 2003

University of St Andrews



Abstract

A text to video instant messaging system has been successfully implemented using Java and the Microsoft Speech API. In particular the RMI functionality of Java has been exploited for communication between the separate sections of the code. The system has been designed and implemented using the software engineering protocol DSDM. Extensive testing has been carried out throughout the implementation of the system to ensure that it is as reliable and failsafe as reasonably possible. This document discusses how the system was created, the achievements and failings of this system and explores what could have been done to extend the project further.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project is 18,003 words long including project specification and plan.

In submitting this project to the University of St Andrews I give permission for it to be made available for use in accordance with the regulations of the university library. I also give permission for the title and abstract to be published and copies of the report to be made and copies of the report to be made and supplied at cost to any bone fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

Contents Page

Introductory Section

Abstract 1

Declaration 2

Contents 3

Introduction 5

Project Details

Summary of Achievements 7

Main Design Considerations 10

Detailed Design Considerations 11

Algorithms and Data Structures 13

Specific Implementation Decisions 15

User Interface Features

Evaluation and Critical Appraisal

Server Classes 23

Client Classes 26

GUI Classes 27

Speech and Video Classes 29

Known Limitations 29

Comparison to Original Objectives 30

Possible Extensions 31

Comparison to Similar Work 32

Conclusions 34

Appendices

Appendix A – Objectives 36

Appendix B – Context Survey, Specification and Plan 37

Appendix C – Interim Report One 57

Appendix D – Interim Report Two **58**
Appendix E – Testing Summary **60**
Appendix F – Status Report **62**
Appendix G – UML **63**
Appendix H – Maintenance Document **68**

Introduction

At the beginning of this academic year, 2002 – 2003, I was given the task of implementing a program with the following requirements:

- Instant messaging functionality
- Text to video functionality
- Multi-platform capability
- Low bandwidth communication
- Completed system with documentation by 23rd April 2003

That is, two users should be able to communicate over a network connection using typed messages. These messages should be used to create a speaking face that says the entered words on a remote user's computer using the Java Speech API (JSAPI), Java3D, and the remote method invocation (RMI) feature of Java. Full implementation in Java ensures, in theory, the portability of the software. The program should be usable on any computer system that has an implementation of the required Java classes and over virtually any speed of network connection.

The following sections outline the success and failure in completing these objectives. The Project Detail section of this report discusses the main achievements of the project as well as the main ideas behind its design. It explains the novel design features used, clarifies implementation decisions and describes the special data structures and algorithms used to implement the programs. It also outlines the features of the graphical user interface. The Evaluation and Critical Appraisal section of the report describes in detail the classes used to create the instant messaging program, explaining where each could have been improved. This section also compares the final outcome of the project to the original objectives and evaluates it with respect to similar work in the public domain including both other chat programs and other methods of creating facial animation.

Once the specification of the system had been drawn up and the requirements worked out in detail a plan was created to aid the successful completion of the project. This plan specified the process model and tools to be used, as well as the timescale for the development of the various sections of the code. This plan, as well as a detailed

description of the specifications, is shown in Appendix B. The main concept behind the plan was to develop the code in several phases. The code was also to be broken up into different sections to make programming clearer and easier, and testing more straightforward and thorough. These divisions were designed to be as separate and as sparsely linked as possible. The four segments are the server, the client, the graphical user interface (GUI) and the speech and facial animation classes. Some sections took longer than expected to complete and some were shorter. The plan, to implement required sections first then add extra features if time allowed, was followed – there are several desired but not required features missing from the final version of the program. Testing was carried out throughout the implementation of the chat system to reduce errors when sections were linked together.

Using this plan a successful instant messaging system that uses the RMI functionality of Java was created. The text to video functionality has also been successfully implemented – a string can be typed and sent by one user and an animated face speaks it on another user's computer screen. A user can enter their own photograph to create images for the video and retrieve another person's images from a remote computer. The user program is accessed through the GUI provided to allow easy use for both novice and experienced users. An administrator runs the server via a command line interface. This behaviour is achieved by linking together the separate sections of the code to allow them to use the functions implemented by the other portions. To register with the program, or to be able to log in and use it, the client section of the code communicates with the server section. For a conversation to occur, two separate clients must communicate with each other. The user interacts with the client section of the code using the GUI. The animation is displayed in the GUI, but started by a remote client passing a string to it.

The program is not multi platform due to the need to use the Microsoft Speech API, limiting my program to Windows machines. There has been limited testing to see whether the program runs over a slow network connection. Some of the features that have been omitted from the final version of the chat system are: the ability of a user to block someone from talking to them; emoticons, such as ☺ have no effect on the image; only two people can be in a conversation at one time. Facilities have been built into the code to allow many of these features to be added later without many problems.

Project Details

Summary of Achievements

The main achievement of this program is, I believe, the successful implementation of a text to video instant messaging system. Comparing what has been achieved to the details set out in the requirements specification indicates that all the main features have been completed but some of the extra functionality is lacking. The programs created make up a text to video instant messenger system that is easy to use and can be run over a relatively slow network connection. A picture chosen by the user appears to speak the words typed. However, this system is not multi-platform and is not completely written in Java as it uses a C header file and a C++ source file to access the MSSAPI methods. It is also missing some of the desired, but not required, features.

The program has two main sections: the server program, which monitors who is online and stores details about each user; and the client program, which is used to connect to the server and to hold conversations between users. The server can be run on any computer which has the Java SDK installed, the clients are limited to use on Windows machines which have the Microsoft Speech API (MSSAPI) installed.

The server end of the system is run using a command line interface and occasionally outputs a list of registered users and a list of those users that are currently online. When the server starts up it reads in information about existing registered users from a text file. Each time a user's details change or a new user registers this file is altered in order to maintain a backup of the information currently held by the server in case it crashes. While the server is running it holds information about each user in User data-structures, which also monitor whether or not a user is still in contact with the server and therefore whether or not their connection has been unexpectedly terminated.

The client program is accessed using a simple GUI that has three main screens. These include the start-up screen, which allows an existing user to log in to the server or a new user to register with the program. The second main screen is the main window, which shows a list of the user's contacts. These are people that the user has decided to add to a list of their friends and acquaintances. People on this list can be invited to participate in a conversation when they are online. This window indicates to the user

whether or not a particular contact is currently online by separating the contacts into two lists. Those people that are online are placed at the top of the window, and those that are offline at the bottom, separated by the word 'Offline'. The menu of this window allows access to the help and information files, the user to log out or exit the program, or to search for and add a new user to their contact list. The final type of window is the conversation window. A copy of this window is displayed on both computers that are involved in a conversation. It is shown when a user clicks on another user's name, to start a chat. This window is made up of three sections: an area where the user types, an area where the conversation so far is displayed and a larger area where the video images are displayed. As many of these chat windows as the user wants can be open at one time although it may prove confusing if several images are speaking at once. However, only two people can be in a conversation at any one time, to avoid confusion. All the windows are as compact as reasonably possible to cause minimum interference to any other programs that may be in use at the same time. Access to the help files is through the menu at the top of the various windows. The help file is also available through the web page at index.html in the home directory of the program. The ability to turn off speech and video in a particular chat window has not been implemented. Also, users cannot remove a contact from their list. Users cannot block people form talking to them. This would have been a nice extra feature but time was short and this was not required for the program to work.

The text to speech section of the program uses the MSSAPI, rather than the JSAPI. This is because on detailed inspection, both the FreeTTS and IBM ViaVoice implementations did not have the functionality required for lifelike animation. The JSAPI does not provide methods that allow enough information to be extracted from the timing of the sounds being processed to create realistic animation, only a list of phonemes for a given word can be found. For this reason the system has been limited to use on Windows machines using the MSSAPI, which does allow access to this detailed timing information. The string typed by the user is passed to the MSSAPI using a native interface, which gives access to the methods in a C++ class that uses functions interfaced by the MSSAPI to generate speech. Periodically the current outputted phoneme is checked and the correct picture for this sound displayed. Doing this rapidly enough creates the illusion of a talking face. The images are displayed at virtually the same time as the corresponding sound is spoken, so the rapidly changing

image appears to be speaking in time with the voice. This is linked to the instant messaging section of the code so that text typed by the user is animated on the receiving user's computer. No time was available to create many of the extra features described in the specification, such as facial expressions, nodding or blinking. Also the original plan, which aimed to use Java3D to morph between successive photographs, was not required. A quicker and simpler method, which produces convincing results, was used instead. This method just flicks between images, as in a standard cartoon, rather than interpolating between them. The result of using this simpler method seems as realistic as using the morphing technique. Another advantage of using this simpler method is increased portability. Java3D is not available for as many platforms as the Java Foundation classes that I have used in my implementation.

The multi-platform capability was required to allow the program to be used on as many different platforms as possible, to allow as wide a range of users access to the system as possible. This was to be achieved by the exclusive use of Java, which would have allowed the program to be used on any computer where the required sections of Java had been installed. Unfortunately this was not achieved due to the use of the Microsoft speech API. Using a JSAPI implementation would not have extended the range of platforms extensively anyway, as there are no alternative implementations than for Windows and Linux.

The system was required to work over a slow connection. I have tested the program running two clients on the same computer connected to the Internet via a slow connection, with the server running remotely. This works successfully, showing RMI methods can be called over a slow connection. As network communication is done using RMI calls the program should not have too many problems when run over a slow connection. Problems may occur when transferring images between users, though the pictures are all fairly small to minimize problems here.

The predicted timescales for each section were slightly inaccurate. The creation of the instant messenger program and GUI was much more complex and took significantly longer to program than originally anticipated. In fact, this section of the program forms the bulk of the code written. Working out how to write some of the sections

was quite difficult, especially working out how conversations between users could be started up and continued. This was because the order in which the stages were to occur using RMI was difficult to work out. Less time than anticipated was required to implement the video sections of the code, partly due to the change from Java3D morphing to switching between images, which created an equally acceptable result. In fact, the more rapid rendering of this method may have increased the performance of this section of the code. Morphing between images takes significantly more computing power than simply flicking between images. As the video needs to be produced in real time the rendering time is a more important consideration than a slight improvement in picture quality.

Main Design Considerations

The main idea behind the design of this text to video instant messaging program was to make it as accessible to as many people as possible. Existing video chatting programs are inaccessible to many users due to the extra hardware necessary to use them, the fast connection required and the limited range of platforms that the programs are implemented for. The user end program was to be as simple as possible to use. People do not wish to spend hours reading through a user manual before they start using a program. The use of my program is fairly intuitive and the basics are very simple to pick up. The animation was to be as lifelike as possible, but only require one photograph of the user. The entry method for this photograph should be as simple as possible to use.

Another thought behind the design was to ensure that there was some kind of working product by the deadline. The use of the DSDM model, which developed the product in several increasingly complex working stages, resulted in a product, which although missing some of desired features works successfully and can be easily used for its intended purpose. Important sections were created first, with desired extra features added later. The program was designed so that it was made up of several modules that fitted together. First, a text based chat program was created and tested then speech and video were added. Detailed commenting throughout each stage of the implementation aided testing and integration as the intended purpose of what the code should have done was clear.

Detailed Design Considerations

The original concept behind the design of the structure of the code was to make it as modular as possible. That is, the code was to be grouped into collections of classes that each carried out a particular role in the functionality of the code. Each section should run almost independently from the other sections. This type of code structure is particularly useful when following a cyclic process model to create the system. It makes new sections easier to add, and simplifies the replacement of existing modules with more complex code. Modular code also makes it easier to adapt to requirement changes or to add other improvements to the code.

My program was built up in sections that were integrated in stages to create a working instant messaging system. The four main sections of the code are the server, the client, the GUI and the speech and animation classes. There is also a group of classes devoted to creating the different images used for the video from the user's original photograph, but as I did not write these they are discussed only briefly. These sections are all linked together to create a functioning system, but the links between them are as minimal as possible, or through defined interfaces, to allow easy replacement of sections.

The functions of the server section are as follows. The server allows users to register with the service. It monitors who is currently online, keeps track of user details in User data structures and records each user's current IP address so that conversations can be started between users. Conversations do not pass through the server but directly between clients. The server does not contact the clients, instead the clients access certain specified server methods using the remote method invocation functionality of Java. The remotely accessible functions are:

- Register
- Find and add a new contact
- Remove a contact
- Log in and out of the server
- Start a conversation with an online user
- Add a new person to an existing chat

The server also has some private functionality, including creating the security policy to monitor who has access to what on the host computer, checking whether clients are still connected, and ensuring that users are aware of who is currently online.

The remotely callable methods are defined by the ChatServer interface, which extends java.rmi.Remote. The extension of this class allows the methods defined in the interface to be called by remote computers connected to the computer running the server. I have chosen to use the default RMI port 1099 for simplicity. I decided to use RMI to create my instant messaging program, partly because it was something new to learn. However, the main reasons for choosing RMI were that it eliminated the need for a communications protocol to be established and solved any problems caused by a firewall being installed in the testing environment. If the port 1099 is blocked by the firewall then RMI uses IP packets to carry messages. The port that these pass through is unlikely to be blocked.

The functionality of the client section of the code is as follows. The client calls the accessible RMI functions of the server to carry out some of its functionality, such as the user logging in, or finding out information about another user. It acts like both a server and client when a conversation is occurring – another client can call this client's methods and vice versa. It passes on all messages to the other users in a chat, as well as displaying the conversation so far. It uses the text to speech section of the code to create and display a moving image representing what has been typed. The client is bound to the server on a well-known IP address. This means that the server should always be run on the same IP address, so that the distributed clients are able to connect to it. By binding to the remote server and creating an instance of the ChatServer interface each client is able to access the RMI methods contained in the Server class. The class ClientHost creates an instance of the Client class for the user that is bound to the RMI Registry to allow other users to access the methods implemented there.

The third section of the code deals with generating the graphical user interface, which is described in detail in the User Interface Features section of this report.

The final section of the code deals with creating and displaying the video image and sound corresponding to the typed text. This is done using two Java classes, MSSAPI and JFacePane. A C header class and a C++ class are also involved. The text to speech section of the code converts the entered text into speech. This is done as soon as possible after the text has been sent, and in sync with the displayed video image. The image generating code only uses a single original photograph of the user. Transforms are applied to this photograph to generate a series of images representing the different sounds. When a conversation occurs the required image is displayed. After a short time period this current displayed image is altered to represent the current sound. The animation is in sequence with the generated speech.

It was not necessary to morph between images as expected in the plan, as I found that simply switching rapidly between images produced a convincing animation. Also the timing and sounds did not need to be stored as the name of the current phoneme can be accessed directly using the MSSAPI, and the correct image for this phoneme displayed. If a sound occurs for longer than 20ms, which is the refresh rate of the video, it is simply displayed again.

The basic UML for my project is shown in Appendix G. The first four diagrams show the classes involved in the sections of the code – the server classes, the client classes, the GUI classes, which include the text to video code, and an overview of the classes used to create the visemes from a single user photograph. The final diagram, split over two sides shows the linking between the sections. Although there appear to be a lot of connections between sections, most of the links between areas of code run between only a few classes, for example to the server. This is to be expected as the Server holds a lot of the functions that the client code uses, so needs to be linked to for access to these functions.

Algorithms and Data Structures

There are two main data structures used in my project to store information about each user. These are User and ClientUser.

A User data structure holds all the information about each user for the server including the full name of the user, the name they wish to be displayed to other users, their date of birth stored as dd/mm/yyyy, the allocated id number of the user, a Vector of the id numbers of other users that are on this person's contact list and a Vector of users that have this person on their contact list. There are also values that record the current IP address of the user, which may change every time they log in. This value is set to null if the user is not logged in. There are also two Boolean values that record whether a person is logged in and whether the user has checked in recently. The empty constructor for this class creates a null person where all the values are set to null or equivalent. The non-empty constructor sets the values for the name, date of birth of the user, the name to be displayed (shown name), id number and IP address, which is null until the user logs in. The Vectors are initialised to empty Vectors, checked in is set to true and logged in is set to false. A set of methods follow the constructors to allow access to the values stored, while leaving the values themselves private so that they cannot be accidentally changed by another section of the program that gets a copy of the variable.

The method `IsOnline(String address)` sets the IP address of the user to the string provided when a user logs in. It also sets the Booleans indicating that the user has logged in and has checked in to true. A thread is started that sets checked in to false every seven seconds. `IsOffline()` changes the values to indicate that the user is now offline. This method does not stop the thread running as the checked in flag will eventually be set to false, to doubly ensure that the person is logged out. If the logged in values are at any time accidentally set to true somewhere in the program, this thread will ensure that the user does not stay incorrectly logged in for too long.

The rest of the data structure is involved in verifying that the user is still in contact with the server. This is done so that if the user's connection is accidentally lost they can be removed from the list of online users without the need for the logout function to be called by the user, who would be unable to do this. Every four seconds each client calls the checkIn method of the server. This then calls the correct checkIn method for that user. Calling this User method stops and resets the timer that will, after seven second set the check in value to false, if it is ever eventually allowed to do this.

The client side equivalent of the User data-structure is the ClientUser. This data-structure is simpler than the server version and only allows for the storage and retrieval of information, not for its modification. It also omits the check-in code and IP address storage, which is requested from the server as it is needed, that is, when a conversation is to be started. As the server method to start a conversation had been defined I decided to use it to return the user's IP address. Also the IP address is liable to change if a user goes offline and then comes back online, due to the dynamically allocated IP addresses used by most dial-up ISP's. This method of locating the IP address only when it is needed ensures that the correct IP address is used to start up a conversation between two users. A second, simpler, data-structure was implemented, rather than just requesting the required information from the server each time it was needed, as requesting the data once and storing it locally takes up fewer resources than repeatedly requesting the user information over an network connection that may be slow. It also reduces the work of the server.

Specific Implementation Decisions

There are several sections of the code that were set up at the beginning of the implementation to allow extra features to be more easily added at a later stage of development. Some of these features have not been implemented, either due to lack of time or they were deemed unnecessary. However, the initial code for these sections still remains. This code allows for the features to be added more easily at a later date if required. Some of this code is described in the following paragraphs.

The method `addToChat(int my_id, int their_id)` has not been implemented. The purpose of this method was to allow an extra user to be added to an existing conversation. I decided to omit this section of code after I had worked out how to do the animation. The animation code finds what the current viseme should be and chooses the correct picture to be displayed. If two people are speaking to a third person at the same time then two different sounds may be said at the same time. This would lead to the wrong face being shown for a particular sound and maybe other errors in the creation of the sound. Also, having two faces, possibly talking at once, would confuse the user. Finally, the significantly increased complexity of adding this

section would have taken a long time to code and I felt that there were more important sections of the code to focus on.

Each User holds a list of people that user has added to their contact list. It also holds a list of users that have this user on their contact list. This was originally done so that a person could be informed when someone added them to their contact list, and they could reciprocate or block them as desired. This functionality was not implemented but the basic structure exists for this to be built into the program.

There are some sections of code that do not appear to have been implemented in the simplest manner possible. The simple methods in server, such as getName(), which jut return a value stored by the server, had to be included as simply passing a user's data structure to a client using RMI caused synchronization problems. It was also simpler and quicker to pass the short strings returned by these methods than to return a large User data structure. Finally, allowing user programs access to the User data structures has the potential for misuse by the client programs, and the data structure could become corrupted. A HashTable was used to store the conversation windows so that more than one chat window could be open at one time, and the correct chat window could still be updated. Each conversation is allocated a number by the starting user. The open window is then hashed using this number, which is the key for the conversation. Both parties involved in a conversation use the same key so that the right conversation can be referred to when calling the RMI methods. For the RMI security manager to be started an existing security policy needs to be in place. This is done by placing a policy file granting access to all users in the directory of the program. When the server is started it is run with a command to use this security policy. This is then immediately overwritten by the RMI policy, to prevent unauthorised access. There is also functionality in the code to log in another user. This was part of the testing for the program – it was quicker to log two or more people in from the same version of the program than it was to run multiple copies of the program. The program was later tested on separate computers to ensure that this did not cause a problem.

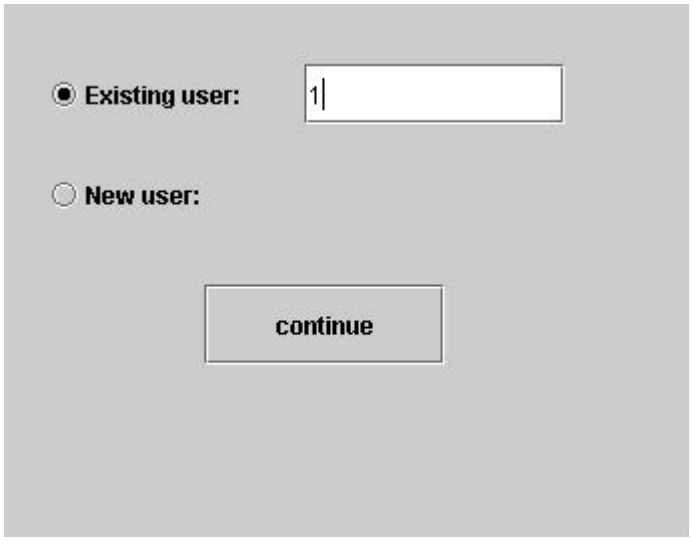
Finally, some way of checking if users still have a connection to the server needed to be found. The server is not able to contact clients; communication can only go in the other direction, with clients contacting the server. I made the decisions to allow only one-way traffic to make the implementation of the server simpler. This problem was overcome by having the clients call a check in function periodically to indicate to the server that they are still connected and running successfully. This is done by calling the `checkIn(int id)` RMI method which sets a flag in the user's data structure, indicating that the user is still online. The private class WhoIsOnline extends Thread and runs as a separate thread to the main path of the program of the server class. This thread periodically prints out who is currently online and a list of registered users. However, its main purpose is to find out whether each user is still connected to the service. It does this by checking the checked in flag in each User structure every five seconds. This flag will be set to true if the user has checked in with the program within the last seven seconds. If the user has not checked in within the previous seven seconds this flag is set to false to indicate to the server, next time the values are checked, that the user has lost contact with the program. If the user has lost contact with the program they are logged out by the server and removed from the list of people currently online.

User Interface Features

There are three main classes involved in creating the GUI: StartUpWindow, OnOfflineWindow and AChatWindow. They were designed to make the program as easy to use as possible for the user. They were also designed with the intention of allowing as little scope for user initiated error as possible, whether intentional or accidental. There are few places in the GUI that allow a user to enter malicious data in an attempt to crash the program. This lessens the amount of error checking that needs to be done by the server, so reducing the chance of the program crashing.

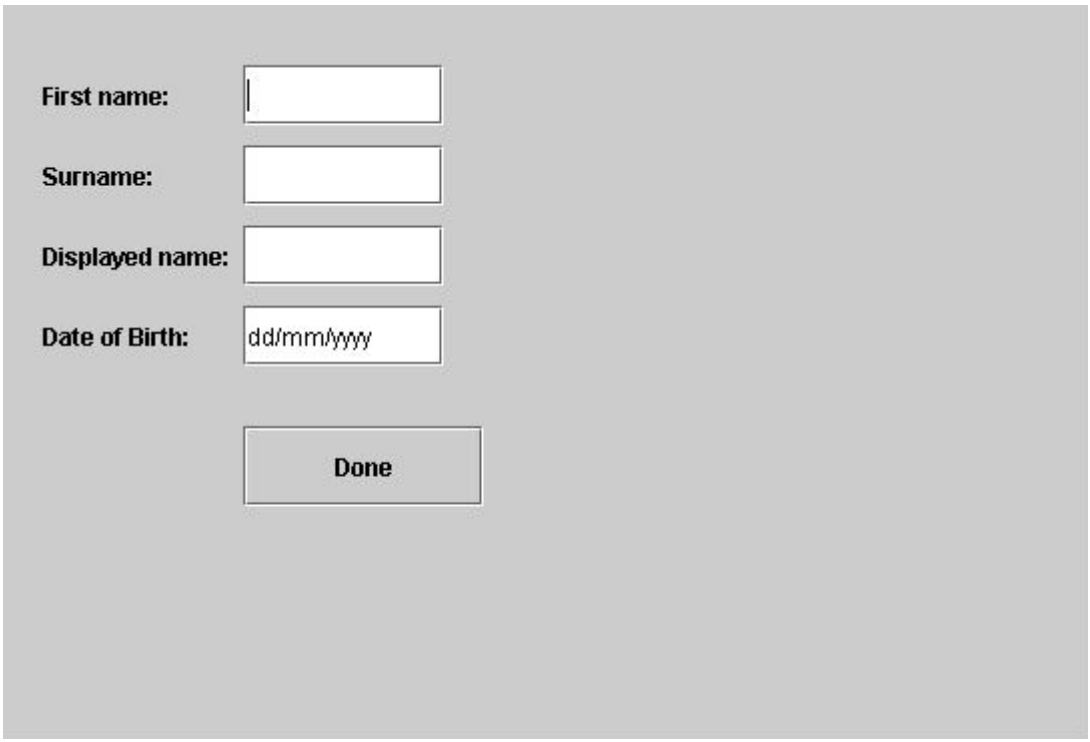
The StartUpWindow is displayed when the client side program is first started up. It allows an existing user to enter their id number and log into the server, or for a new user to register with the service. If an existing user tries to enter a invalid id number, the text field clears and they have the chance to re-enter the number. There is no scope for the user to cause error in this window as even if the user enters malicious

text the id number will just be rejected. A user could log on as another user, either on purpose or by accident. Implementing a password entry scheme could prevent this. Log in is done on id number as this value is guaranteed to be unique to each user. If a valid id number is entered the user is logged into the server using the id number and the IP address of the computer. If the user id falls out of range, or the user is already logged in then a null value is returned and the user gets another chance to try to log in. The StartUpWindow is shown below:

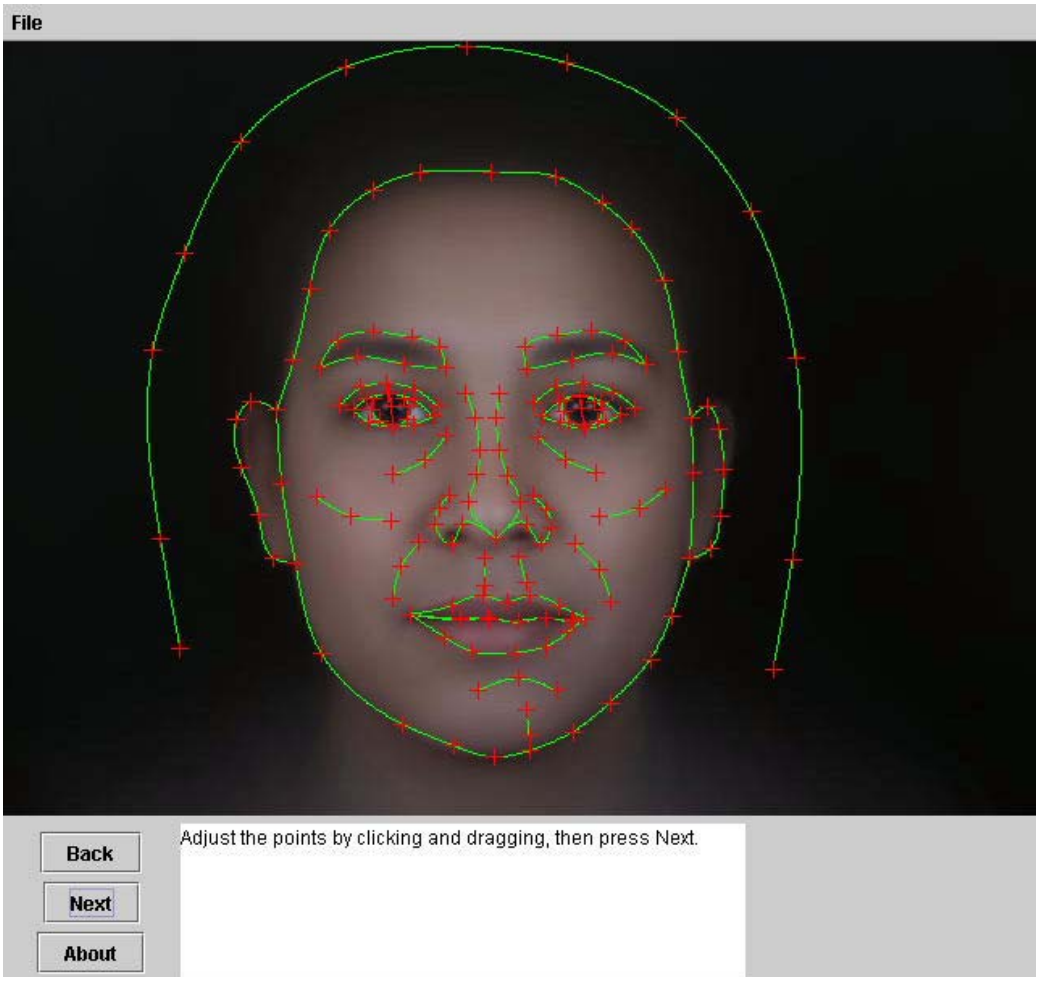


If the user has not used the service before they can chose to register by selecting the Register radio-button. This takes them to another window that allows them to enter their details. The information required is the user's full name, the name they wish to display to other users and their date of birth. If a field is left empty then the empty string is used for that value. This will only cause a problem if the displayed name is left blank – to solve this, the user's id number is allocated to this value if it has been left blank. Leaving the other fields blank will only cause problems for other users searching for this person. In fact, this would be a good way to remain anonymous to other users.

The registration window is shown below:



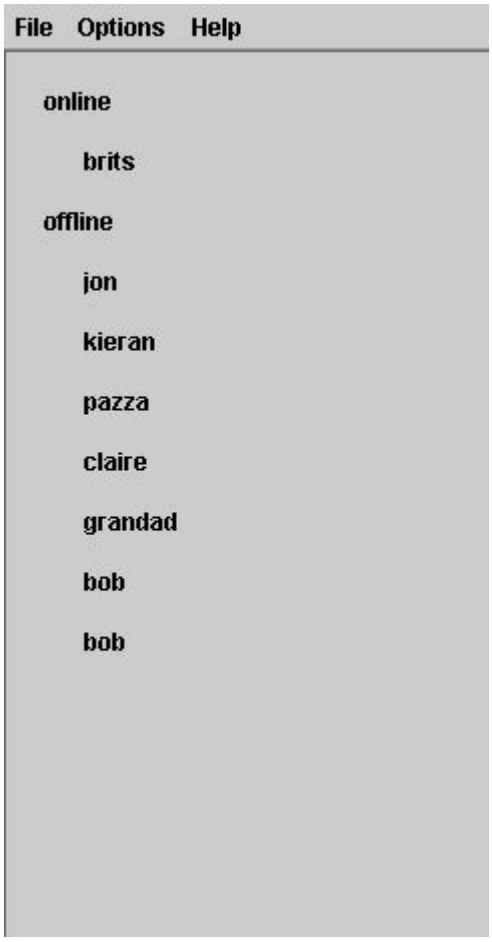
Once the user has registered with the system they are shown their id number then asked to enter their photograph for the viseme creator. This section of the code is slightly adapted from an existing program that performs transforms on images of faces. The user has to move points onto their face to indicate the shape:



More precise positioning is done around the mouth in order to get the best images possible, ready for the video section. The seventeen viseme images are then created and saved. Other users can access these images at the start of a conversation so that a video of the correct face is displayed.

The OnOfflineWindow class creates the main window of the program. It is used to indicate to the user who is currently online and who is currently offline of the people on this user's contact list. The window shows this by splitting the contact list into two, with the online users displayed at the top of the screen and the offline users under them. The online users each have a mouse listener associated with them, so that when their name is clicked on, a conversation with that person can be started. The contents of this window are periodically updated in order to maintain an up to date list, as users log in and out of the system. The menus for this window allow the user to exit the program, log out, add a new user to their contact list, access the help files and find out

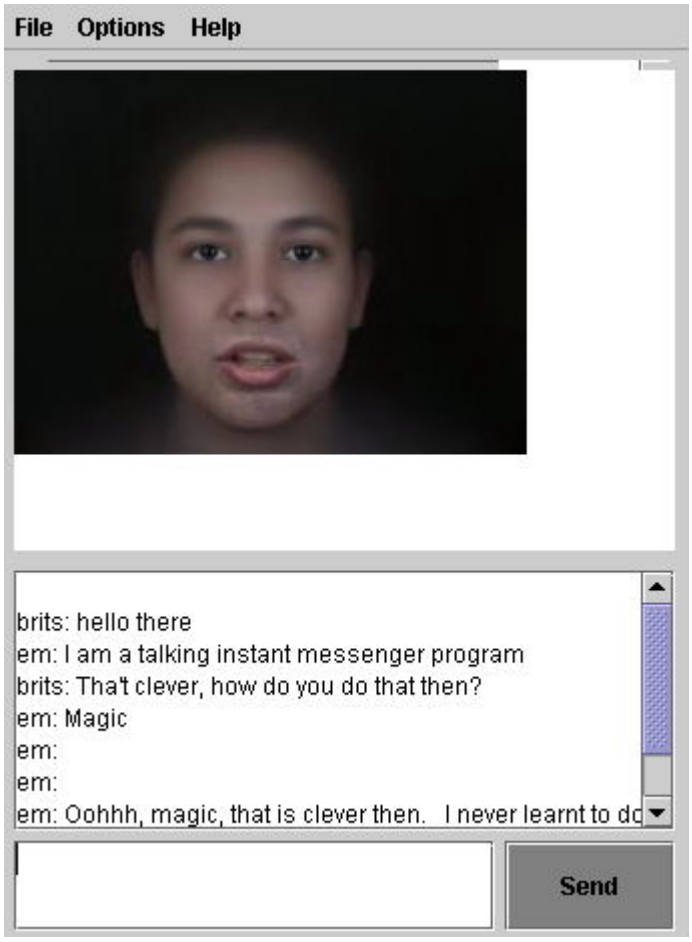
about the author and program version. The user below has built up their contact list, displayed in this OnOfflineWindow:



The private class AddContact, which is accessed from the OnOffLine window, creates a window that allows the user to search for and then add a new user to their contact list. The user can search on full name, displayed name or date of birth. Once one or more of these fields have been completed, a list of possible matches is returned from the server by calling the findUser server method with the entered parameters. The user can then select which person to add to their contact list from the list offered to them. If no one is suitable, they can search again with different values.

The final main GUI class is the AchatWindow class. This is the window that is displayed when a conversation occurs between two users. It is made up of three panels and a menu bar. The menu provides access to the help and documentation files, as well as the facility to leave the chat. The three panels are for text entered by the

user, the conversation carried out so far and a larger panel for the video image. Text typed by the user is sent to the other chatter when the send button is pressed. The conversation so far window of both users is updated with the relevant text, as well as the display name of the user that sent the text. On the remote user's computer the image is made to speak this user's text, and the sound from the words is played. The following image shows the user speaking in this window:


File Options Help
brits: hello there
em: I am a talking instant messenger program
brits: That clever, how do you do that then?
em: Magic
em:
em:
em: Oohhh, magic, that is clever then. I never learnt to do
Send

Evaluation and Critical Appraisal

The Server

The following paragraphs describe in detail how the server section of the system works. The server side classes allow access to the system, maintain information about users and allow interaction between clients, by providing information to start conversations and allowing users to add another person to their contact list. The server classes, as well as their links to other sections of the code are shown in Appendix G.

The server is implemented in the following classes: ChatServer, which is an interface describing what methods are available to remote users; Server, which implements this interface and contains most of the code for the server side classes; ServerHost, which contains the main method to start the server; and User, which is a data structure to hold each user's information.

The RMI methods outlined by the ChatServer interface are implemented by the Server class and are described below. `register(String name, String shownName, String dob, String address)` uses the information provided to create a User data structure for that person, which is then added to the list of users. The entered information, including id number, is also appended to the file used by the server as a non-volatile backup in case the server crashes. This file is read in when the server is restarted. The id number allocated to that user is returned once the user has been successfully added.

Once a user has registered with the service, to use the facilities again they must log in. This is done using the `login(int id, String address)` method that returns a Vector of the users that are currently online and on that person's contact list. The IP address is required to start conversations between users. A check is made to ensure the id number passed to the server is valid and falls within the current range or id numbers, which is incremented each time a new user is added to the program. This ensures that all id numbers are unique. If the id number is not valid of the user is already online, an empty vector is returned and no contacts are visible to the user. If the id number is valid the user is added to the list of people who are currently online

and their IP address is set in their User structure. If the user wishes to disconnect from the service they need to indicate this to the server by calling the `logout(int id)` method. This method removes the user from the list of online users and sets the person's IP address to null, indicating that they are currently offline.

The `checkIn(int id)` function returns a Vector of the user's contacts that are currently online, replacing the list provided when the user logged in. It also sets the checked in Boolean flag in this person's User data structure. This ensures that the user is not logged out when they are still connected to the service.

The two functions `findUser(String name, String shownName, String dob)` and `addContact(int my_id, int their_id)` allow a new user to be added to a person's contact list. Firstly, findUser is called to get the id numbers of possible matches to the criteria entered. The searching user enters one or more of the following: full name, displayed name and date of birth. findUser then checks each of the non-empty fields passed to it against the corresponding field of each User in the list of registered users. The id of each possible match is then added to a Vector of possible candidates if it is not already present. This is returned so that the user can select the required person. The id number of the selected person is then returned to the server using the addContact method.

Once the User structures have been altered to take account of this new information the file holding the user information is updated. As Java provides no simple method for inserting into a particular point in a file, the data file is read in a line at a time, until the id number at the beginning of the line matched the id number of the user adding a contact. The id number is delimited by a ':' so that the whole number is easily read. The id number of the contact being added is then inserted at the end of the list of contacts. As each line is read it is written to a temporary file in its original state. The line to be altered is written to the temporary file in its new form. The contents of the data file are then overwritten with the contents of the temporary file, and the data file is updated. Altering the file by first creating a backup also reduces the risk of data loss if the server crashed mid way through the write. The original file will either be intact, or the temporary file will be intact, so there is always a backup. The `removeContact(int my_id, int their_id)` has been partially

implemented. It alters the User data structures, but does not alter the data file, so I have left this method inaccessible to the user – there is no way to call it using the GUI.

The method `startChat(int my_id, int their_id)` is called by a user program when the user wishes to start a chat with another person. This method returns the current IP address of the other user's computer. This enables the requesting user to access the other user's RMI methods and so start and participate in a conversation.

The remaining RMI methods implemented in the Server class are `getMyContacts(int id)`, `getOnContacts(int id)`, `getName(int id)` and `getShownName(int id)`. These are simple methods that just return the information requested. That is, the user's full name, the name the user wishes to display to other people and the two types of contact list for a user.

The Server class also contains some private functions. When the server first starts up it needs to read in information on existing users, stored in a text based data file. From the information in this the Server class creates User data structures for each user. Each user is added to a Vector that records the registered users. Users that are currently online are also stored in a Vector recording online users. The private method `getUserStruct(int id)` returns the User data structure for a given id number. It does this by searching though the Vector of registered users and comparing the id number of each user to the required id number and returning the user that matches. The method `myOnline(User this_user)` returns a Vector of id numbers, stored as strings, of the people on this user's contact list that are currently online. It compares the id number of each user on the contact list to each user in the online list and adds the id number of each one that matches to the list of online users, as a string as integers cannot be stored in Vectors. I could also have used the Integer class here.

The class ServerHost contains the main method that is called to start the server. When this occurs the security manager is started, which controls who can access which files on the host computer, and prevents malicious access. Once the security manager has been started an instance of the Server class is created. The Server class sets up the user information and defines the methods that can be called remotely, as described

above. The IP address of the host computer is then found and bound to this computer's RMI Registry

The Client

The client section of the code provides the main, non GUI-based functionality of the client side code. It provides the means for communication between clients, as is required during a conversation. The following paragraphs describe this code in detail, but an overview is available in Appendix G.

The client section of the code is made up of the following classes: ChatClient, which is an interface defining the client methods that can be called remotely; Client, which implements these methods; ClientUser, which is the client side equivalent of the User data structure; ClientHost, which binds the program to the RMI Registry; and WindowTest which contains the main method for the client side classes.

The interface ChatClient defines the client methods that can be called by remote users. These are `displayMessage(String message, int id)`, which displays the text entered by the user on the other user's display (ie the one whose method this is) `startConv(String address, int my_id)`, which starts a conversation on this remote computer `addToConv(String address1, int id1, String address2, int id2)`, which adds this user to an existing conversation and `endConversation(int conv_id)`, which ends an existing.

The class Client implements the following methods: the method startConv binds this client to the remote computer accessing its methods so that this computer can access the remote clients's RMI methods, and so participate in a conversation. The chat id number counter is then incremented to give the next conversation a different key. This is to identify chat windows if more than one window is open at one time. A ChatWindow is created and hashed using the counter value as the key. The key for this window is then returned to the calling user. The local method that corresponds to the RMI method startConv is startConvLocal(ChatClient c, int my_id, int their_id, String my_address, ChatServer the_server). This initiates a conversation by first calling the remote user's startConv method. The conversation id number returned by

this method is then allocated to a local ChatWindow, which is displayed and added to this user's HashTable of current ChatWindows. The RMI method displayMessage updates the text in this user's chat window when called by a remote user. It gets the correct chat window from the HashTable and calls its updateText(String text, int who) method. The int is a flag to indicate that a remote user has called the update method. A remote user calls the RMI method endConversation when they wish to end the conversation. The message "User ended conversation" is displayed for three seconds, after which the window is closed. The client also holds a reference to the server so that the server's RMI methods are available to it.

Graphical User Interface

The GUI provides a simple and easy means of communication between the user and the main code of the program, contained in the client and server sections of the code. There are three main classes associated with creating the GUI StarUpWindow, OnOfflineWindow and AchatWindow. The use and appearance of the GUI is described in the Novel User Interface Features section above. This section further describes the operation of the code. The flow of control between sections of the GUI is shown in Appendix G. The connections between the GUI code and the rest of the system are also indicated.

When the client side program is first started, after a ChatServer has been made and the code bound to the server's RMI Registry a StartUpWindow is created. The ChatServer is passed to this so that the RMI methods can be called. Once the layout of the StartUpWindow has been set up the user is able to either enter their id number if they have one, or register as a new user. Once the user has logged in and the Vector of contacts received the name and shown name of the user are retrieved from the server and a new ClientUser is created for this person. The contact lists for this user are then retrieved from the server and each person has a ClientUser data structure created for them. The contents of the online list, returned by the server when to user logs in are then turned into ClientUsers as well. This is repeating what has already been done and should be coded by referring to the local data structures in the full list of contacts. Once all this has occurred an OnOfflineWindow is created and displayed.

The OnOfflineWindow class contains most of the functionality of the client side code. It holds an instance of Chat Server so that the server's RMI methods can be accessed, in particular to find information about the users on this person's contact list from their id number, such as when a conversation is started. Once the window layout has been set up the id number of the user is found from the user data structure passed as a parameter. A ClientHost is then called to create a Client and bind its methods to the local RMI Registry. The check in facility of the server is then called to ensure that the user is not inappropriately disconnected. Vectors are set up to maintain lists of who is currently on and offline of that user's contacts. The Vector of online users is returned by the server when the user first logs in, or checks in to the server. The offline list is created by finding out who is in the contact list, but not on the online list. The `setupWindow(Vector online, Vector offline)` method is then called, which returns a JscrollPane containing a list of all the contacts, separated so that the top half of the screen contains the online users, and the bottom half the offline users. Each online name is allocated a mouse listener, which monitors when the name is clicked on. When this happens a conversation is started between this user and the user whose name had been clicked by calling the startChat server method to get the IP address of the user. A ChatClient is the created so that the other user's RMI methods can be accessed, and a conversation started.

The UpdateThread is started, which periodically redraws this OnOfflineWindow with updated information on who is online, returned by the server each time the user checks in. The check in is also carried out by this thread, every four seconds. The Vector of online users returned by the server when this user checks in is made up of id number stored as strings. To be useful to this user they need to be turned into data structures. The program does this by requesting the information about each user from the server each time a new list is retrieved. This is very inefficient. It would have been better to find as much information as possible about the online users from the existing information stored by the client. RMI calls take time, especially if they are done over a slow connection, whereas finding the information from a local machine is much quicker. Also, if all the users are constantly asking the server for information the server will start to run slowly. This is not currently a problem for my system, as there are not very many registered users, and not many people are online at one time.

When a user's name is clicked on and a conversation started an AChatWindow is displayed on each user's computer. When a user has entered text the method updateText, contained in this class, is called to update the text in the conversation window. One of its parameters is a flag indicating whether a local or remote user has called the method, so that the appropriate name can be displayed next to the added text. This also indicate whether or not the image should be made to speak the text – only a remote user updating the text should cause this to happen as there is no point in repeating to the user what they have just typed.

Generation of Speech and Facial Animation

Four classes are involved in the generation of the speech and facial animation: JFacePane, which retrieves the current sound and displays the appropriate image; MSSAPI.java, which provides a native interface to access the code in the MSSAPI, which is not written in Java. The two non-Java classes are MSSAPI.c, which is a header file automatically generated from the native interface code in MSSAPI.java, and MSSAPI.cpp, which implements the methods outlined in the header file and allows access to the functionality of the MSSAPI.

The video generation is much simpler than originally anticipated. A JPanel is set up to contain the image and then every 20ms the image contained in this JPanel is updated to represent the current sound being processed. When no sound is being said, the image is still repeatedly changed, but between the same plain face. The current sound is found by calling the getViseme() method of MSSAPI, which in turn calls the equivalent method in the C++ code which generates the speech using the functionality of the MSSAPI.

Known Limitations

There are not many known bugs in this program; most of the code seems to work as expected. The main problem is that so much time was spent creating successful code that some of the requirements have been omitted.

One problem with the initial specification, rather than a bug in the code is that anyone can log in as any other user – there is no password protection. This could be simply

implemented in the logging in screen when the program first starts up. When a user first registers they would be required to enter a password. This would then have to be stored somewhere on the server.

If a client stops running it is just logged out of the server and the user just needs to re login. However, it is not always apparent to the user when an error has occurred. The GUI virtually always continues to run if an exception has been thrown. To solve this problem the user just needs to log out then log back into the program. More problems are caused if the server computer is accidentally reset, as happened once. All the client programs need to re-login once the server has been restarted, but there is no way of informing the users that this needs to be done.

A problem that occurs occasionally is the server logging someone out for no reason. This may be because a problem has occurred with the RMI communications, or because the timings are incorrect for the check-in code, but this does not happen often enough to easily fix. One final problem is that if a user has more than one conversation open at one time, if a message is sent to one conversation, all the other faces speak the words, even though the typed message is not displayed in that window. This does not occur if two separate versions of the program are running on the same computer at one time. If this occurs the second phrase to be sent is queued until the first sentence has been completed. Another problem is that when a new user first registers with the program, they need to log out and log back in again to be recognised by the system.

Comparison to Original Objectives

The main body of code has been successfully completed but some of the additional features have not been fully implemented. These include the ability to have three people in a chat at one time. This was omitted as it would have taken a long time to implement, and I was running out of time in that cycle of development. It was more beneficial to the project to move on to the next section of development, and come back to this if time was available. I also thought that having only two people in one conversation makes what is happening much clearer. Having more faces speaking could prove confusing. I have also not had chance to implement a limit on the number

of conversations that can occur at one time. Again I thought it beneficial to move on to other more crucial elements of the project.

The functionality to block a user from starting a conversation with another user has not been implemented, and neither has the ability to remove a user from your contact list. These functions would be useful if another user was abusive or offensive, to prevent the user making further contact. It would also have been useful for a check to be made when a user wishes to add someone to his or her contact list that that person wants to be added to that contact list. An extension to this could be to allow a user to reject a conversation if they do not wish to talk to the requesting user. Just just shutting the conversation window can already do this.

Another feature that has been omitted is the ability of the user to turn off the video or speech in a particular chat window. This would be useful if a user was participating in several conversations and one was particularly important. The rest could be turned to silent to enable the user to concentrate on the important conversation.

Possible Extensions

Although I have successfully implemented a text to video instant messaging system, there are many extensions that could be made to it, above those described in the specification that have not been achieved, discussed above. These include making the GUI clearer, better looking, more functional and less dull. At the moment the colours are all default shades, mainly grey. The appearance of the GUI could be significantly improved by adding a little colour and some graphics to make the system more appealing to a wider audience. Keeping the GUI simple, however, is still important.

Making the program simpler to log into by allocating each user a unique username, or entry using email address, would make the program easier to user. It is very easy to forget an id number, especially if the program is not used for a period of time. Remembering a username or email address is much easier, as it relates to the person, rather than just being a random number.

The talking face could be improved. Although it would probably be unnecessary to morph between images as described in the original plan, the animation could be improved in other ways. These include the addition of facial expressions, random head movement, blinking and the nodding and shaking of the head. Adding features like these will all add to the realism of the computer generated video. Another nice feature would be the ability to resize the talking face to the desired dimensions. This would be particularly useful if the video was to be shown to a group of people or if the user has poor eyesight. Enlarging the image would be particularly beneficial to this group of users. A twist on this could be to add sound effects. For example, the voice could be raised if the user wishes to shout, or lowered to whisper. When a ☺ is entered a laughing sound could be produced. As far as I know, the MSSAPI only has limited functionality to do this.

An extension to the concept of this text to video instant messenger would be to add speech recognition capability. Rather than relying on a user to type their text into the chat window, a user could choose to speak into a microphone. The words would then need to be translated to text, sent using RMI as for the existing program to reduce bandwidth requirements, and then converted to speech in the usual manner.

This idea could also be extended to develop a very low bandwidth video conferencing program. The speech recognition software that I use is probably not good enough to do this yet, but allowing several people to be in a conversation, each with a microphone could in the future provide low bandwidth communication in this manner. Even allowing several people to be in a conversation using the existing program, with each user typing their contribution to a discussion, could partially emulate a video conferencing program. Features such as transfer of files between programs or even a simple whiteboard facility could be added to improve the range of functions available through the program. One possible use of a program like this could be distance learning, especially when there is no high bandwidth link available.

Comparison to Similar Work

Many chat programs already exist. Some of these are discussed in the context survey in Appendix B. Most of the existing programs are text-to-text instant messengers,

such as MSN (Appendix B [1]) or ICQ (Appendix B [4]). My program has the same basic functionality as these programs, but is much simpler. It is missing most of the fancy features of these programs, such as file transfer or sharing, sound effects or use of a whiteboard. However, my program does have the same basic uses, that is, it allows communication over the Internet. Some more recent programs use what has been typed to manipulate an image. For example IMPersonna (Appendix B [7]) moves a cartoon face to make it appear to be saying what you have typed. This is quite similar to my program, but my talking face is more realistic. Another program with more lifelike animation is the SeeStorm (Appendix B [6]) instant messenger. This requires a microphone to use it, and does not allow a user to enter text using a keyboard. The level of animation is good, but it is difficult for a user to enter their own image for manipulation, they need to use one of the provided faces. Programs such as NetMeeting require a fast connection, cameras and microphones. They are much more realistic than my program as they use real images for the video, but are available to significantly fewer people due to the set up costs.

I have limited access to the results of other types of animation techniques such as geometric facial animation (Appendix B [14]) or physics and anatomy based models (Appendix B [16]). However, I believe that I have chosen the simplest method for the fairly realistic results obtained. This program is accessible to any user who can put a photograph on their computer. None of the other methods provide the simplicity or merely altering a few points on a face to generate a real time, fairly realistic video.

Conclusions

The original aim of this project was to create a text to video instant messenger system written entirely in Java that would run over a slow network connection. Although not all the features originally specified have been successfully implemented, I believe that an acceptable final product has been created. This text to video instant messenger can be used to participate in a video conversation, but still has areas that need developing.

The main achievements of the project include the successful completion by the required deadline of a text to video instant messenger system. This program operates over two or more computers to allow users in separate locations to communicate with each other. This communication is done by each user typing text at one end of the conversation that is then displayed and converted into speech and video at the other end. The video is based on a single photograph of the user. A user can log in to the server, and remain logged in as long as they retain a connection. A user can also log out. A new user can register to use the system, and an existing user can search for and then add new users to the list of people they can talk to. Help files are accessible to aid the user with the running of the program, though this should not be necessary as it is simple and intuitive to use.

The system also has several limitations, where areas specified in the requirements document have been omitted, generally due to time constraints. These include the ability to add users to a conversation, block another user or remove them from a contact list. Several of the desired features have also been omitted, including the use of facial expressions.

Writing a comparatively large piece of software is always challenging, even more so when the author has limited experience in this area. In particular, sufficient testing of what had been written was a particular challenge as even slight changes in code could lead to unpredictable behaviour. Ensuring that every return from a method call was the expected value, and not null was also fairly difficult, and I do not believe that either of these tasks has been carried out fully. Testing a distributed application was more taxing than I originally anticipated due to the differing behaviour of code when it is run on a local host or over an Internet connection. The use of RMI, which uses the

TCP/IP stack of a computer, even when the method calls are to a machine running the same IP address or local host should have prevented this, but didn't.

Even though problems were encountered during the implementation of this system a working solution was created. This was due to the careful planning of possible solutions to all conceivable problems as well as following the original plan. The timescale for this ensured that a lot of work was done early on in development. The incremental development also ensured a working final solution.

The system could be improved to become a viable releasable product, although work would need to be carried out to ensure that the program was more robust, more elegant and more efficient, to deal with a large numbers of user. It could also be made more pleasing to look at. However, the software in its current form could still be released as an acceptable and unusual product.

Appendices

Appendix A - Objectives

The aim of the project is to create a text to video instant messaging program. The video will be composed of an audio interpretation of the inputted text and an image of the person speaking which moves its mouth to match the speech. The video will be created by performing transformations on a single image of the user. The program will be written entirely in the Java programming language in an effort to make it as multi-platform as possible.

Instant messaging allows users to communicate in real-time over a network by typing messages. Most of the current messaging programs use text only, although some more recent ones allow for the use of speaking animated characters or use speech from a microphone to manipulate an image and are Microsoft based.

The first aim of my project is to create a simple text based instant messaging program that allows several conversations to occur at once and also for several people to be in one conversation at the same time. The next objective will be to add sound using the FreeTTS implementation of the Java speech API. This library turns typed words into speech. Currently this implementation only provides a simple male voice but if time allows more voices could be added. Finally, moving images will be added to the program by manipulating the photograph of the user using Java 3D. The required shape of the mouth will be found by splitting up the speech into its constituent phonemes (sounds), which have to be mapped to the appropriate visemes (face shapes). The timing of each sound needs to be found before the appropriate transformation is applied to the original image to create the correct mouth shape at the correct time and for the correct duration. Applying this to a series of sounds creates a sequence of images and animates the photograph.

If time allows more features will be added such as blinking or breathing, head movement, either controlled by the mouse or random, and facial expressions for the emoticons that are often used in chat rooms to show the mood of the user.

Appendix B - Context Survey, Specification and Plan for the Design and Implementation of a Text to Video Instant Messaging System

Contents

Title		
Contents		**37**
Project Definition		**38**
Objectives		**38**
Context Survey	**Chat Programs**	**39**
	Text to Speech	**40**
	Facial Animation	**41**
Requirements Specification	**Functional**	**43**
	Non-Functional	**44**
Design	**Sever**	**44**
	Client	**46**
	Text to Speech	**46**
	Image Generation	**47**
	GUI	**48**
Plan		
	Process Model	**49**
	Risks, Constraints and Quality Control	**50**
	Resources	**51**
Appendix 1	**GUI**	**53**
Appendix 2	**Table of Risks**	**53**
Appendix 3	**Gantt Chart**	**54**
Appendix 4	**High Level Design**	**55**
Appendix 5	**Networking overview**	**55**
References		**56**

Problem Definition

The aim of this project is to create a multi-platform text to video instant messaging program using the Java programming language. An instant messaging program allows users to communicate, usually with text only, over a network connection. My program will extend this idea by adding sound and video to the conversation while still allowing communication to occur over a low bandwidth. A realistic talking image of each user will be created from a single photograph of the person. Transformations will be performed on this image to mimic natural facial movements and to give the impression the photograph is speaking. This moving image will be played in time with an audio interpretation of the entered text to make it appear that the user is receiving a video of the other person talking.

There are three main objectives to be fulfilled to create the desired program.

- Design and build a simple text based messaging program. This program should be capable of holding several separate conversations at one time, as well as allowing several users to be in a single conversation. It should also have an easy to use and simple GUI that does not interfere with the use of other programs. If time allows further features, such as altering user status (for example allowing other users to be blocked or declaring an away status) could be implemented.
- Convert the text to speech using a text to speech implementation of the Java Speech API (JSAPI). This is to be done on the receiving computer, rather than on the server or sending computer, to decrease the amount of information that needs to be sent over the network and so allow the program to be run over lower bandwidth connections.
- Find the phoneme timing information using the text to speech engine and use this to animate the facial image. To do this animation appropriate transforms need to be applied to the original picture to generate images that correspond to the different phonemes. When these are morphed together and played in sequence they give the impression that the face is speaking. If time allows, extra movements could be added to the animation such as blinking, head movements or expressions derived from emotions.

Context Survey

Chat Programs

Many chat programs already exist, with the current standard for communication being text to text. There are two main types of chat program – stand-alone programs which allow messaging in a separate application and chat rooms, which run in an Internet browser window. The second type, the chat rooms, show a list of who is logged into the chat room and a text box containing all the text that has been written by all users. This usually proves to be quite confusing as no one is sure who is talking to whom or in what order. These chat rooms can, however, be run on any platform as they run in a browser window. Using this style of chat application to design my text to video instant messaging program would prove very confusing, as many faces would be speaking many different conversations at once and saying things that are not relevant to a particular user.

The four main programs available for online conversation are Microsoft [1], AOL [2] and Yahoo [3] instant messengers and ICQ [4]. These programs have a list of people you have selected to chat to. When any of these people come online their name is highlighted and a conversation can be started. They all allow several conversations to be held at once or several people to be in one conversation at one time. This type of application is more suited to the program that I wish to write, as there are fewer people involved in a conversation – typically the limit is around four. The existing programs are written originally for the windows operating system, although ports to other systems have been made. They also only allow for text conversations. Another method for real-time communication is video conferencing using programs such as Microsoft NetMeeting [5]. However these require a fast connection and a microphone and camera, which many people do not own. Using text to video for my chat room overcomes these problems.

More recent chat programs extend the existing instant messengers. A company called SeeStorm [6] has created quite a realistic looking speech to video program. It allows one user to speak using a microphone and uses this speech to create a talking face and shoulders on another user's computer using only a 28.8k connection. A user can even provide an image of their own face for the video, which can be made to appear to talk,

do random head movement and facial expressions based emotions entered by the user. However, this program has some disadvantages - a microphone is needed to make it work, which not many people own, there is no textual representation of the conversation, only one two person conversation can occur at a particular time and the program is only available for Microsoft Windows. Another Windows program called IMPersona [7], which extends Microsoft Instant Messenger has been also been developed. IMPersona provides a talking face animated from the text entered by the user. Only one end needs a copy of the program – the other user can just use their unmodified Microsoft Messenger, and it also runs over a 28.8k connection. It responds to emotions, causing the image to smile or frown as desired and provides cartoon faces as well as lifelike human faces. However, the GUI is poor. It takes up most of the screen, has a tabbed pane for multiple conversations so they are difficult to track and half the window is filled with advertisements resulting in a very cluttered look. The animation is also of quite a poor standard with the heads having no shoulders, the random movement is excessive and the voices are of a poor quality.

Text To Speech

My text to video instant messaging program will use the FreeTTS [8] implementation of the JSAPI [9]. There is not yet an official Sun implementation of this API so a decision had to be made about which third party implementation was to be used. There are many existing text to speech APIs available although most are not written in Java. One of the most widely used is Microsoft Text-to-Speech [10], which has good documentation, a wide range of features and has several voices to chose from. However it was created for windows so is not multi-platform and it is not designed to adhere to the JSAPI. There are several implementations of the JSAPI in existence. These include IBM's Speech for Java [11], which runs in Windows and Linux Redhat. It has a short trial period before it needs to be paid for and it is built on top of IBM's ViaVoice, which also needs to be bought. Also, the implementation is incomplete, it has undergone only limited testing and it doesn't run on the Java Virtual Machine. Another, full, implementation has been written by Cloud Garden [12]. However this only runs on the Windows platform and also needs to be purchased. FreeTTS, as the name implies, is free. It is also open source and has extensive documentation so should be fairly simple to learn to use. It is written entirely in Java, but is based on Flite, which in turn is based on the Festival Speech Synthesis System, which is written

in C++ but has initial JSAPI support. FreeTTS has Windows, Macintosh and Unix implementations so is the most multi-platform of the available implementations. Unfortunately it only has a partial implementation of the JSAPI. In particular, speech recognition has not been implemented, but as this is not going to be used in my project this is not a significant problem. There are also some methods missing for finding timing information of the speech. This is more of a problem, as this information is required to do the image manipulation. However, the lower level functions are accessible and it appears that this omission from the implementation should not be a significant problem.

Facial Animation

The last programming section of my project involves generating a lifelike talking face. There have already been many attempts to generate realistic but artificially generated facial animations using various different methods. These are successful to varying extents due to the complexity of the problem – there are many different aspects to a realistic animation including general facial movement, blinking and eye direction and correct mouth movements. When people speak the rest of the face does not remain static, non-verbal signals such as the eyebrows moving up and down, wrinkles appearing and expressions to convey the exact meaning of what is said are also used – these expressions are linked to what is being said [13] and can even aid in speech comprehension. The seven expressions in the Ekman set are happy, sad, anger, disgust, surprise, fear and neutral and these need to be used if the animation is to be made more realistic. Human viewers are very sensitive to inconsistencies in facial movements, which is why until very recently computer generated cartoons, such as Toy Story, have avoided showing humans speaking. Animation can be done in two or three dimensions depending on the resources available and the animation method to be used. There are three main types of animation used to manipulate the facial image and replicate speaking. These are geometric facial animation, which distorts the underlying 3D geometry of the face, physics and anatomy based animation, which model the facial tissue and its elasticity when distorted by muscle action, and image based facial animation, which morphs between real facial images.

Geometric facial animation, as used to do voice puppetry [14], animates faces by performing geometrical transformations on the image [15]. These are either rigid

motions such as jaw rotation about a point when the animation is speaking or non-rigid motions such as that of lips, which are controlled by spline functions. Spline functions ensure that the points being moved interpolates smoothly through all the desired positions. However, using this method requires time and a talented animator to generate lifelike facial expressions. Voice puppetry is one way of controlling what a facial animation says. In [14] a probability distribution of possible facial motions is derived by analysing videos of people speaking. This information is used to train a Hidden Markov Model, which is a statistical model that estimates the most probable viseme for the current sound. It takes into consideration the previous state when producing the current face shape, so co-articulation is taken into account. Co-articulation is the phenomenon where the current face shape is influenced by the previous face shape due to latencies in tissue motion. Other models lose this information as they only use the current viseme to generate the current image.

Physics and anatomy based models model the face in terms of its anatomical structure, that is, the underlying bone, the muscles and the facial tissue, as described in [16]. Applying forces to the ends of the muscles to strain them or displacing the muscle ends moves the face by shifting the surface points. Restorative forces, volume preservation damping and the elasticity of the muscles and tissue are all taken into account to try to model the human face realistically. A force equation is generated taking these factors into account and then integrated over time to calculate the surface movement for the animation. To set up this kind of model a facial model needs to be acquired, for example using a laser scanner. As for the speech puppetry above, this type of model can be trained using video to make it appear more realistic.

Image based animation interpolates between images of the various face shapes made when a person speaks. This can be done in several different ways. A collection of images can be stored in a database, representing the face shapes of all the different sounds. This size of this database can vary from several hundred thousand photographs if triphones are used [17] to just one [18] depending on how the images are selected and used. The largest databases proposed store an image for each triphone, which is a set of audiovisual sequences extracted from, for example, a video of someone speaking. To cover every single possibly combination of sounds would require the capture of many images, as well as the space taken to store them. This

approach has high redundancy – nowhere near this many images need to be stored. Reducing the redundancy can be done by composing a smaller set of visemes and getting photographs that correspond to these, usually from a video where the most extreme mouth shape for a sound is taken to represent that sound [19]. A morphing algorithm then needs to be applied to generate the intermediate images and create the animation. One of the most recent approaches is to acquire a single photograph of the subject with a neutral facial expression and generate the set of images representing the visemes using this. These additional images are created using a set of transforms which have been found by averaging the mouth positions of people speaking a sentence containing the basic set of visemes. The most extreme position for each viseme is averaged and the transform required to generate this position is found. Applying these transforms to the original image generates the required set of mouth positions. These images can then be morphed together, as for the above methods, and played with the sounds that created them to produce a lifelike video. I am using this last method, as it will be the easiest way for a user of the chat program to provide their viseme image.

Requirements Specification

Functional Specification

This project will design and implement a multi-platform text to video instant messaging program written entirely Java. The program should successfully run over a fairly slow Internet connection, for example using a 56k modem. The program will be accessed through a simple graphical user interface that causes minimum interference with the use of other programs. A list of people that can be contacted will be shown in one window and any chats that are started will each occur in their own window. Double clicking on the name of the required person starts a chat. Each chatting window will contain the images of the other people in the conversation, a textual record of the conversation so far and a box to enter text. Up to three conversations can be held at any one time and up to three people can be in one conversation at once. A low number has been selected as allowing any more people than this to speak at once will be confusing and it is unlikely that a typical home computer could generate video for this many conversations at one time. It should be possible to turn off video or speech generation in any of the conversations and just read the text, in case there is too

much information for the user to follow. A user should be able to search for their friends by name or user id then add them to their contact list. This needs to be done by both parties before a chat between the users can start. A user should also be able to remove a person from their contact list, stopping that person from chatting to them – blocking that user. The user should provide a photograph to be used to create the video. If no photograph is available there should be a selection for the user to choose from.

Non-Functional Specification

The deadline for the creation of this product is 4pm on Wednesday 24th April 2003. The project report, software and documentation are all to be delivered on this day. Before this two interim reports are to be completed documenting progress so far. These are to be submitted on Wednesday 4th December 2002 and Wednesday 12th March 2003. A twenty-minute presentation demonstrating the software is to be held on Monday 12th May.

The user manual will be available on-line, accessible as an html document from the menu of the chat program. This will give full details on how to use the product, although program will be simple enough to be fairly self-explanatory. The program is aimed at a wide audience so should be simple to use by both computer novices and by those more experienced.

The program is being written with the 1.4 version of the Java API. This is required for the use of Java3D version 1.3 and FreeTTS version 1.1.1, which are being used for the graphical and text to speech sections of the project. As the project is being written entirely in Java it should run on any computer that has these API's installed.

1 Server

There is to be one instance of the server, which will run on one of the computers in the honours lab, as these computers are permanently hooked up to a network connection. It will monitor who is online, keep track of user details and contact lists and allow people to register with the service. It will also provide the addresses of clients, so that

conversations can be set up between users. Conversations will not go via the server, but directly between clients to reduce the workload of the server and increase the speed of the program. This is shown in Appendix 5

1.1 The server will track

1.1.1. Who is registered in an array of 'Users', ordered by the id number

1.1.1.1. A User is data-structure containing information about each user. More specifically the user's full name and date of birth, the name they wish to use with the program, the id number allocated to them when they first registered, an array of id's of the users on their contact list, an array of users whose contact list this person is on and, if they are online, the address of the computer that instance of the application is being run on.

1.1.2. Who is online, monitored using the id numbers

1.2. The clients will access the server using Java's Remote Method Invocation (RMI). RMI allows methods on remote computers to be called, so in this case the chat clients are able to call some of the server's methods. It also removes the need to create a protocol to manage the data being passed between sockets, establishes the initial connection to the remote computer and solves problems caused by firewalls and proxy servers. There will be several methods available to the clients using RMI. These will allow the users to:

1.2.1. Register with the program. Once the person is registered and has entered some basic details about themselves they will be allocated an id number and their details will be stored on the sever.

1.2.2. Find a registered user. A person needs to add the people they wish to talk to to their contact list. The id number may not be known so this enables them to find this, and then verify the correct person has been found.

1.2.3. Add a contact to their contact list using the id number of that person.

1.2.4. Remove a contact from their contact list.

1.2.5. Login to the server when their client program starts.

1.2.6. Logout of the server when the program is closed.

1.2.7. Start a chat. The address of the person to chat with is held by the server and needs to be passed to the client in order to start the chat, as it does not run via the server.

1.2.8. Add a new person to an existing chat. The address of the new person needs to be passed back to the person requesting the addition.

1.3. There are also several methods that are not to be accessible by other applications. These functions allow the server to ensure that the program functions correctly and include:

1.3.1. Starting the security manager – important when any program can access the methods.

1.3.2. Periodical checks to see if users are still logged in. This is necessary to ensure that if a connection is lost without the logout method being called, for example if a cable is pulled out, the user is shown to be disconnected.

1.3.3. Broadcasting to the appropriate users that a particular person has logged in or logged out, enabling them to indicate to the user whether that person is available to chat to.

2 Client

Each user program is a client. The clients will use the RMI functionality of the server to access its methods. However, once this address is found the clients act like servers and the chat messages are sent directly to the other people in the conversation without the need to use the server. This reduces any bottleneck that would be caused by sending all the messages through the server, increases privacy and hopefully will decrease the waiting time for receiving messages.

2.1 The client will call the available methods on the server, described above.

2.2 The client will also have some responsibility of its own

2.2.1 Rejecting a request for a conversation if there are already three conversations occurring.

2.2.2 Disallowing the addition of another person to an ongoing conversation if the conversation already has three participants.

2.2.3 Sending the typed message to all users in the chat.

2.2.4 Using the GUI to display the messages

2.2.5 Using the text to speech and video functions to create the video and then display it.

3 Text to Speech

The FreeTTS implementation of the Java Speech API will be used to convert the text received by the client into speech. The basic principles of just making the program speak the entered text are fairly simple – the error checking and voice is set up, then

the phrase is passed to a FreeTTS method, which makes the computer speak the words. This however, needs to be extended to allow for the conversion to video.

3.1 The basic speech should occur as soon as possible after the program has received the message, however it must be in sync with the video produced by the graphics module.

3.2 The speech is to be split up into its constituent components, its phonemes, which will be stored along with their timing information.

3.2.1 The start time of each sound, its duration and the sound all need to be stored.

3.3 This information will then be passed to the image generating section of the code

4 Image Generation

4.1 A single photograph of the user is required

4.2 Transforms are to be done on this photograph to generate appropriate images

4.2.1 Transforms to emulate speech are to be done only on the mouth area to reduce computation time. These transforms will be found and used as follows

4.2.1.1 The list of phonemes found using FreeTTS are to be converted to a list of corresponding visemes. This is a many to one mapping, as several sounds have the same mouth shape.

4.2.1.2 To get a sequence of images with the correct mouth shapes the visemes are used to find the correct transform for that particular viseme. This is a one to one mapping.

4.2.2 To mimic facial expressions transforms need to be applied to other areas of the face. For example to the eye area to imitate blinking and whole face transformations to show expressions such as smiling, laughing or frowning.

4.3 Once a sequence of images has been generated they need to be put together to create the speaking animation. This is done as follows

4.3.1 Select the first two images, keeping a record of the duration of the sounds they represent.

4.3.2 Make the first image opaque and place it on top of the second image.

4.3.3 After a short time period, for example 0.05 seconds, alter the image shown so that it is a percentage mixture of the two original images. In this case, as there are 0.8/0.05 = 16 intermediate images to create in this time, 93.75% of the first image and 6.25% of the second image are to be used to create the intermediate picture.

4.3.3.1 Different visemes need to be blended at different rates, for instance consonants have more effect on face shape than vowels. This is known as co-articulation and could be taken into account if time permits by weighting the percentage usage of each picture.

4.3.4 This is repeated at equal time intervals, until the second image is fully visible and the original image is totally transparent, to create a sequence of intermediate images. When played to the user at 0.05 second time intervals the image will appear to be making the appropriate sound.

4.3.5 This process needs to be repeated on all the images making up the words and sentences typed by the user.

4.3.6 When all the images are played in sequence the original photograph will appear to be speaking

4.4 This animation needs to be played in time with the speech generated by FreeTTS in order to create a realistic looking video of the person speaking.

5 GUI

The general design of the main GUI is shown in Appendix 1

5.1 The GUI is to have three different types of window:

5.1.1 The set-up window, which is used when a user first registers. This will ask the user for their name, date of birth, the name they want displayed on the screen and request a forward facing photograph.

5.1.1.1 If a photograph is no available at this time it should be possible to use a supplied photograph and enter the users photograph at a later time

5.1.1.2 If a user's photograph is used the key points need to be found by the user. For example several points around the mouth need to be selected as well as eyes and nose. This screen will be contained in the set-up window.

5.1.2 The contact list window. This will contain the contact list of the user and will not take up a large proportion of the screen.

5.1.2.1 When a contact is online this will be indicated to the user, either by somehow highlighting the name or moving the person to the top of the list.

5.1.3 The chat window. One of these windows will be opened for each conversation that is occurring on the computer. This window will contain three areas:

5.1.3.1 Somewhere for the user to enter their text.

5.1.3.2 A box containing the talking faces, this can be switched off if required.

5.1.3.3 A script of what has been typed so far.

Plan

As with all software engineering projects the design and implementation of this program requires a plan. This section documents how this project is to be completed including which process model is to be used, risk reduction measures, the available resources, the constraints of the project, methods of quality control and when the various sections outlined in the specification are to be completed.

Process Model

The process model that I have chosen to use is based on the Dynamic Systems Development Model (DSDM) [20]. This is a Rapid Applications Development (RAD) Model, which can be used for the creation of pieces of software written using an object-orientated language such as Java. This process basically divides the implementation of the program into cycles of time-blocks. Each block has a set of goals associated with it as well as a set of desirable features. The goals must be completed in the time-block, so these should be realistic, but the desirable features are only to be implemented if time allows. If the implementation of the additional features is not successfully completed they are moved to be goals in the next time block. The aim of using a cyclic pattern like this is to ensure that a usable product is available at the end of the implementation period, even though it may not satisfy all the initial requirements. It is particularly useful when the timescale for developing the full product, or the resources available, are limited. The product should still be usable in the state reached at the end of the final iteration even if it does not satisfy all of the requirements. The extra functionality can be added for release in a later version of the software. Before the start of each new cycle a feasibility study needs to be carried out to decide what can realistically be implemented in the next cycle. At the end of each cycle the program is tested to ensure that it satisfies the user requirements for that section of the coding. This reduces the risk of an inappropriate product being created as well as ensuring unreachable goals are not set, as these are decided as the project runs. The Gantt chart shown in Appendix 3 shows the approximate time allocation for

my project. It is divided into four cycles – implementation of a basic chat program with GUI, speech, image creation and report writing, separated by the horizontal lines. These sections have some overlap with work taken from the next cycle. These are the sections that it would be desirable to code in the preceding cycle to add to its functionality but which are not necessary for its successful function. The Gantt chart is flexible – DSDM will be used to decide before the start of each cycle what is feasible to do in the time-block based on the results of the previous time block. The Christmas and Easter holidays give some buffer time that will allow me to catch up if the project is running behind schedule. Also, the initial goal for the completion of the software is set for nearly a month before the actual deadline. This also provides a buffer period, although hopefully this will not be needed as this time is needed to write the report. The high-level design that has been used to draw up the Gantt chart is shown in Appendix 4.

The program has been designed to allow the use of the iterative development model. The code is to be divided into four sections, GUI, chat program, speech functions and image manipulation. This modularity should allow for easy insertion of each new section of code that is written. For example the initial GUI for the program will be basic, but if time allows a more intricate design could be used. Changing which GUI is used should be very simple to do – only the GUI code needs to be changed, no code involved in any other section of the application. Also the initial application will be text based, then speech will be added, then video. The low coupling will mean that adding each new section of code should have minimal effect on the existing code – the function call to, for example, 'MakeVideo' will at first return a blank screen, then play the sound and finally play the full video, without having to alter the code in the chat section of the program. This should make debugging the program simpler, as each section will be working with minimal bugs before it is added to the rest of the code.

Risks, Constraints and Quality Control

The risks to the project, and solutions to any problems that may occur, need to be considered before the start of programming in order to ensure a more successful solution. Appendix 2 shows a table of these risks and possible solutions. The use of DSDM, especially the feasibility study preceding each cycle, should help to reduce risks, as will adhering to the plan. The worst-case scenario is that I am unable to work

on the project. If this occurs the result of the previous iteration will act as a backup. The Gantt chart in Appendix 3 shows the four milestones – the completion of the basic chat program and its user interface, the integration of text to speech and the splitting up the constituent sounds to find the timing, the development and integration of the video, and the completion of the documentation and report. The successful completion of each milestone helps to ensure the project is running on time and increases the chances of a successful end product. Clear and thorough documentation will also help in the understanding and testing of the code. Each function should have a concise but clear description of the intended purpose. Comment layout and code style should be done so that the code is clear, readable and concise. Code documentation should be done in step with coding.

The constraints imposed on the project include the timescale for development (just over 6 months), little previous use of RMI, FreeTTS and Java3D, practical work form other modules interfering with the adherence to the plan, only one person working on the project and possible unavailability of my supervisor. To reduce the risk of failure, testing will be carried out throughout the implementation of the instant messaging program. Each section will be thoroughly tested before it is integrated with the existing program, using a bottom up approach. The various sections of the code can be tested using test harnesses – for example a skeleton main program to check the functionality of a new part of the code, or to see how the whole program runs. For example, instead of manually registering several users each time the program needs to be tested, the details would be included in and entered by the main method to decrease the time taken to do the testing.

Resources

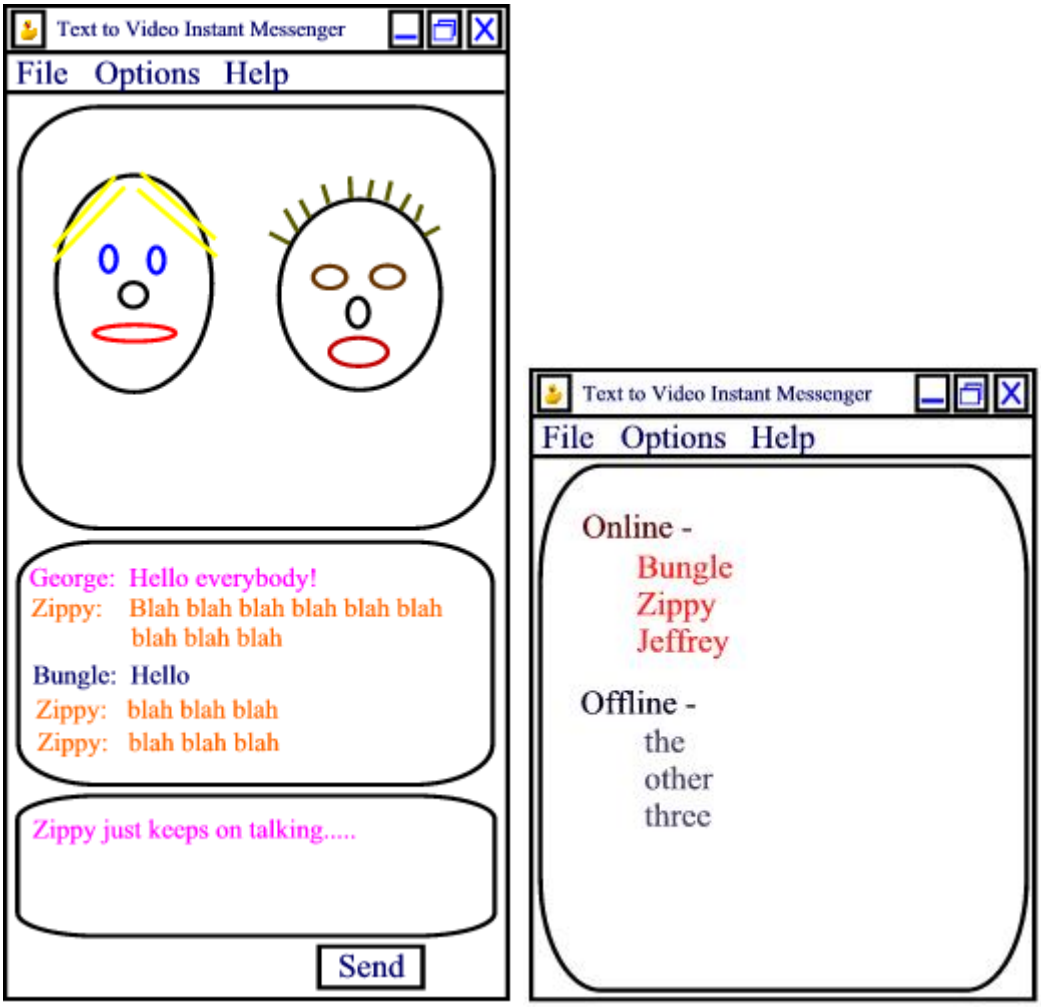
The program is to be written on a 1.6GHz P4 with 256MB of RAM running windows XP Professional, which connects to the Internet at up to 44kbps (with a 56k modem). The server is to be run on one of the computers in the senior honours lab. The computers in this lab are 1GHz P3's connected by a 100Mbps Ethernet, running Linux Redhat 7.2 and Windows 2000 These computers will be used for testing the program, as well as the machines in the first year lab, which run Windows 2000. The wider the range of computers the program is tested on the better but it will at least run on my computer, the Linux machines in the honours lab, and the windows 2000 machines in

the first year lab. The lab machines have a fast Internet connection – this is not typical of many home computers so it will be tested over slower connections. The program is to be written using the 1.4 version of the JDK, using the Java3D and FreeTTS extensions. Two IDE's are available for use, Sun One Studio 4 and Together 6.0. The javac compiler is also available for use if the IDE's prove to be unsatisfactory. Microsoft Office is available for the writing of reports, as is Open Office, both of which are suitable for this kind of work.

In order to keep track of changes to the program a version control program will be used. A logbook is also to be kept documenting meetings, reasons for decisions and general progress, serving as a reminder of why things were implemented in a specific way. Both these methods of recording how the development is progressing will hopefully aid the writing of the documentation and report. When each milestone is reached, this working version of the program will be set as a baseline. This means that a copy will be stored in the version control's archives and can be consulted to aid testing or returned to if unfixable bugs are introduced to the program. Keeping baseline versions of the code will ensure that not everything is lost if a section of the code is somehow lost or deleted. Programs for version control that are available are CVS and the version control facilities of Together. In case of loss of the copies of the current version, backups will be made and kept in a separate location, for example on cd or on a computer in the honours lab.

The drawing up of this specification and plan should ensure the success of the project. Specifying what is required by the instant messaging system and how these requirements are to be implemented before the commencing of programming reduces the chances of the project overrunning, producing an incorrect product or failing to deliver a working end program.

Appendix 1 – GUI



Appendix 2 – Table of Risks

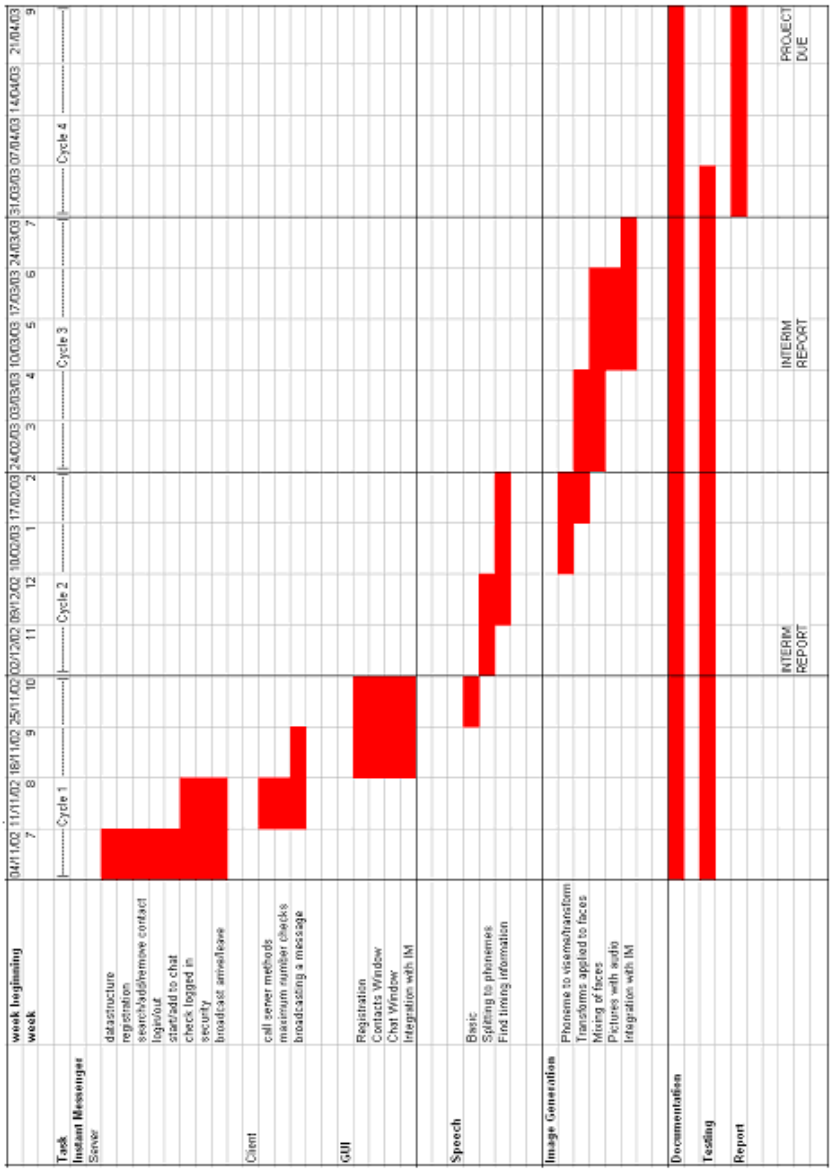
Risk	When	Probability	Resolution
shortage of time for completion of project	Throughout project, but increasing with time	High - this always happens as there is always more to do	Stick to the plan and don't leave everything to the last minute
spending too much time on one area of the code	Throughout the project - it is easy to get sidetracked an not stick to the plan if one part proves especially complex or interesting	Medium	Use and follow the requirements specification and plan
focusing on an unimportant area of the code	Throughout the project - coding of something that is not necessary to the final project or trying to use the wrong solution to a problem	Medium	Use and follow the requirements specification and plan
creation of inappropriate system	Throughout the project but especially at the beginning of each cycle, when the functionality to be added is decided	Medium	Use and follow the requirements specification and plan
lack of progress	At the beginning of the project - it is easy to think there is plenty of time left whereas the schedule for finishing the program is tight	Medium	Start on time
exceeding time available	At the end of the project - it is tempting to try to fix one last bug or add some new functionality when the time should be spent writing the report	High - I tend to be a perfectionist	Stop when to plan says to stop - very few extra marks will be gained by doing this
crisis management	Toward the end of the project - if it is running behind schedule it is tempting to spend long hours trying to satisfy the original requirements	High - it is always tempting to get a program working successfully even when there is not enough time	Set a time at night to stop working - mistakes are made when you are tired
collision of work with other modules	Throughout the project but especially towards the end of semesters when there are more deadlines	Low - these deadlines are known well in advance and are set to avoid collisions	Check the deadlines when they are released and plan work around these
illness	Throughout the project but with more serious consequences towards the end	Low - I'm in good health :)	Get enough sleep, eat healthily and hope.

Appendix 3 – Gantt Chart showing Milestones – Project Monitoring

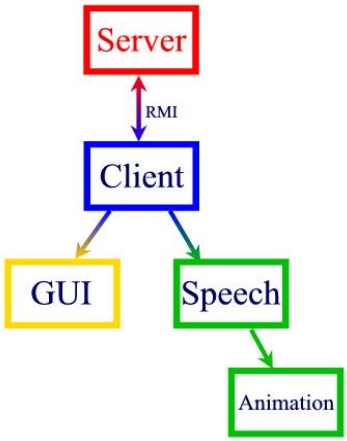
Tasks Completed

	week beginning	23/09/2002	30/09/2002	07/10/2002	14/10/2002	21/10/2002	28/10/2002
	week	1	2	3	4	5	6
Task							
Get list of available projects							
Decide on project							
Aims and Objectives							
Research							
Specification and Plan							
DEADLINES		BRIEFING	WORKSHOP	DESCRIPTION AND OBJECTIVES			SPEC AND PLAN

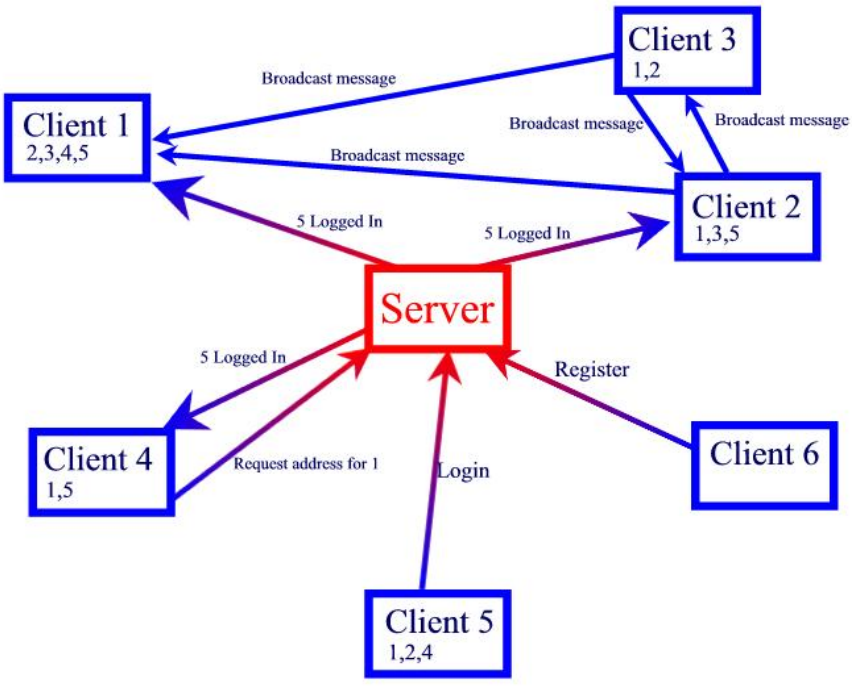
Tasks to do



Appendix 4 – High Level Design



Appendix 5 – Networking Overview



References

[1] http://messenger.msn.co.uk 29/10/02

[2] http://www.newaol.com/aim/netscape/adb00.html 29/10/02

[3] http://messenger.yahoo.com/ 29/10/02

[4] http://web.icq.com/ 29/10/02

[5] http://www.microsoft.com/windows/netmeeting/ 29/10/02

[6] http://ssm.seestorm.com/ 29/10/02

[7] http://www.impersona.com/ 29/10/02

[8] http://freetts.sourceforge.net/docs/index.php 29/10/02

[9] http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-doc/index.html 29/10/02

[10] http://www.microsoft.com/speech/techinfo/apioverview/ 29/10/02

[11] http://www.alphaworks.ibm.com/tech/speech 29/10/02

[12] http://www.cloudgarden.com/JSAPI/index.html 29/10/02

[13] E.K. Walther, Lip-reading, Nelson Hall Inc, Chicago, 1982

[14] M. Brand, Voice Puppetry, SIGGRAPH99 Conference proceedings, 1999, pp. 21 - 28

[15] J. Noh, U. Neuman, Talking Faces, International Conference on Multimedia, 2000, pp 627 - 630

[16] K. Waters, A Muscle Model for Animating Three-Dimensional Facial Expressions, SIGGRAPH87 Conference proceedings, 1987, pp 17 - 24

[17] C. Bregler, M. Corell and M. Slaney, Video Rewrite Driving Visual Speech with Audio, SIGGRAPH97 Conference proceedings, 1997

[18] B. Tiddeman, and D Perrett, Prototyping and Transforming Visemes for Animated Speech, 2002

[19] T. Ezzat and T. Poggio, Visual Speech Synthesis by Morphing Visemes, International Journal of Computer Vision, 2000, Vol. 38, No1, pp 45 - 57

[20] http://www.dsdm.org/en/default.asp 29/10/02

Appendix C - Interim Report One

This report documents my progress so far in the implementation of the text to video instant messaging system that I am to create for my senior honours project. The time scale for this implementation is shown in the Gantt chart in Appendix 3 of my specification and plan. This chart shows that by this point in the year I should have completed the first cycle of programming – the implementation and testing of a basic chat program. That is, the code to do the control of the chatting and the user interface should have been completed.

The actual progress made so far is slightly behind schedule. All the methods have been outlined for the client and server sections using interfaces. The user data structure has been completed, as have the basic functions including registering and logging in or out. No progress has so far been made with the GUI. This means that the project is running approximately two and a half week behind schedule. This delay in progress is in part due to the amount of work that has been set for other modules aggravated by limited access to the computer lab, and four days lost through illness.

With the work I have done so far, it has become apparent that using the Remote Method Invocation feature of Java to create the chat program will be more complex than originally anticipated, particularly in transmitting messages between chatting clients. Working out the problems encountered has also been a factor in putting the project behind schedule. I believe that I have overcome most of these problems and that the project can successfully be completed using RMI.

I aim to catch up on the lost time during the last week of term, when there is significantly less work, and during the Christmas holidays, which I had originally left out of the time plan. This omission was done on purpose in order to enable me to catch up on work that was behind schedule. There are approximately eight weeks left out of the Gantt chart over the Christmas period and although we do have exams during this period I fully expect the implementation to be running either on or ahead of schedule by week one of next semester. This should leave plenty of time to successfully complete the project in the period allowed.

Appendix D - Interim Report Two

This report documents my progress so far in the implementation of the text to video instant messaging system that I am to create for my senior honours project. The time scale for this implementation is shown in the Gantt chart in Appendix 3 of my specification and plan. This chart shows that by this point in the year I should started to do the morphing between faces and integrating the various sections of the chat program. Documentation and testing should also be well under way.

The progress made so far is slightly ahead of schedule. I have a working instant messenger program that speaks and animates a photograph. People can log in and log out, search for and add a new user to their contact list. A new user can register with the program and add contacts to their list. People can be accidentally disconnected and the server deals successfully with this, logging them out automatically after a short period of time. A user can hold as many conversations as they like with other users on their contact list. Only two people can be in any conversation at one time as any more than this would be too confusing, as they may end up speaking at the same time. When a conversation occurs the text typed is turned to speech on the recipient's computer. The sounds of this speech are extracted, as is their timing. Which viseme is being said at a particular time is then found and the correct image for that sound displayed. The image is updated once every 20 ms. The code is almost fully javadoced and inline commented.

It was not necessary to use Java3D to do the animation; a realistic animation can be achieved without blurring between the images. That is the animation is done like a cartoon swapping between pictures quickly enough to make it appear as if it is moving. Another change to the original plan was the requirement to use the Microsoft Speech API, limiting the project to Windows machined. The Java speech API did not provide a mechanism to extract enough information about the timing of the different sounds to provide sufficiently realistic speech. A less realistic model could use the more basic information available through the speech API if the program is required to be multi platform.

There are a few small sections left to do – the help files need to be written and linked to from the program, the program needs to install a security policy file, and I need to test it on a wider range of machines. Extensions to the project could include allowing a user to use their own photograph and adding facial expressions.

Appendix E - Testing Summary

The code was written in sections, which were progressively put together after each one had been compiled and as thoroughly tested as was possible.

The basic User data structure was implemented first, along with the basic RMI functions of the server. The first stage was to make sure that a user could log in and out of the program successfully. This was tested in several stages, firstly over a local connection using the local host, then over an Internet connection running the server and client on the same computer, and then running the server on a remote computer. Once the fact that RMI actually worked had been established, and the code written to make it work with the basic functions log in and log out more functionality was added, including the registration of a new user.

So that I did not have to re-enter the details of several users every time the server crashed or a modification to the code was made I created a simple text file to store information about some users. The server reads this in each time it is restarted. This was originally just intended for testing but I discovered that it was very useful to have a backup of the information stored in a running server, in case it crashed accidentally and the data stored in User data structures was lost.

Once the basic functions had been implemented I created parts of the GUI. This was done to aid testing. It is a lot easier to enter information correctly in a GUI than it is to enter it via a command line interface. It is also easier to see what has gone wrong. Once the basic GUI had been set up I continued to add functionality to both the server, and then writing the code to allow the client to use this functionality. As each method was added it was carefully tested to ensure that it worked as expected. Most of the testing was carried out using the server running on the local host, as this was quicker and easier than connecting my computer to the Internet each time I wanted to test something. At various intermediate stages the code was checked to ensure that it worked when the server was running on a remote computer.

One part of the code that required particularly vigorous testing was the check in threads running on both the client and server. These had to be very carefully checked

to ensure that they did what I expected them to. That is, to log out a user if their connection was unexpectedly lost, but to leave them logged in if they were still in contact. I thought this worked originally, but more thorough testing showed that the server sometimes logged out users when they were still online. This was due to the timing of the checks being slightly out of synch, and was easily fixed by slightly modifying the times.

Once most of the functions had been implemented the main method of testing was to use the program as much as possible. I tried entering invalid values in various places, and fixed problems that occurred. The program was run on several different computers to see if this affected the program and any problems that occurred fixed, to the extent of my knowledge.

Appendix F - Status Report

Summary of objectives met:

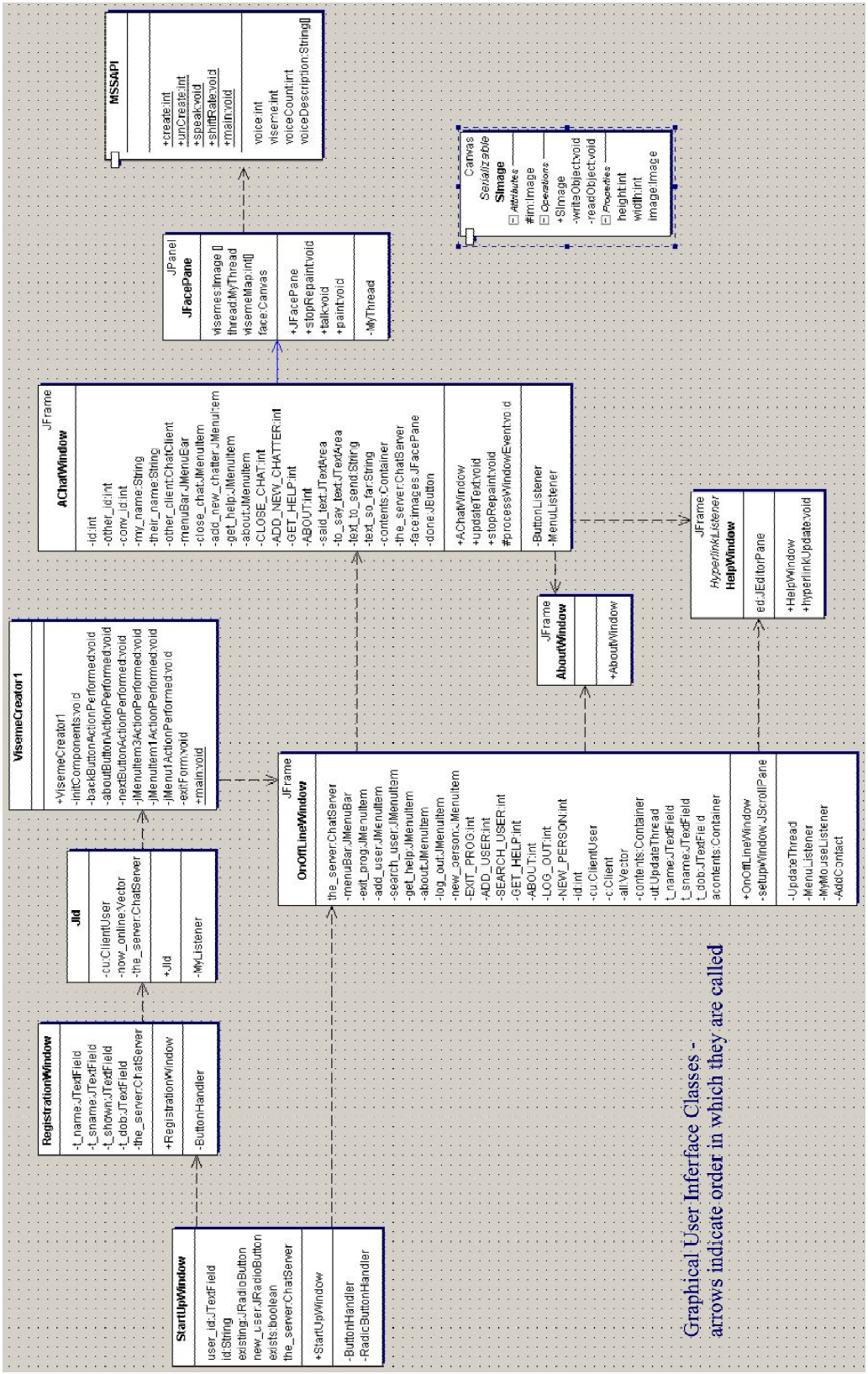
- Successful text to video instant messenger program
    - Instant messaging capability.
    - Converts text to speech.
    - Uses speech to generate video.
    - A user can enter their own photograph or use a default image for use in the animation.
    - A user can retrieve another person's image for use in the animation.
    - A user can search for a person registered with the system.
    - A user can add another person to their contact list.
    - Help files are available.

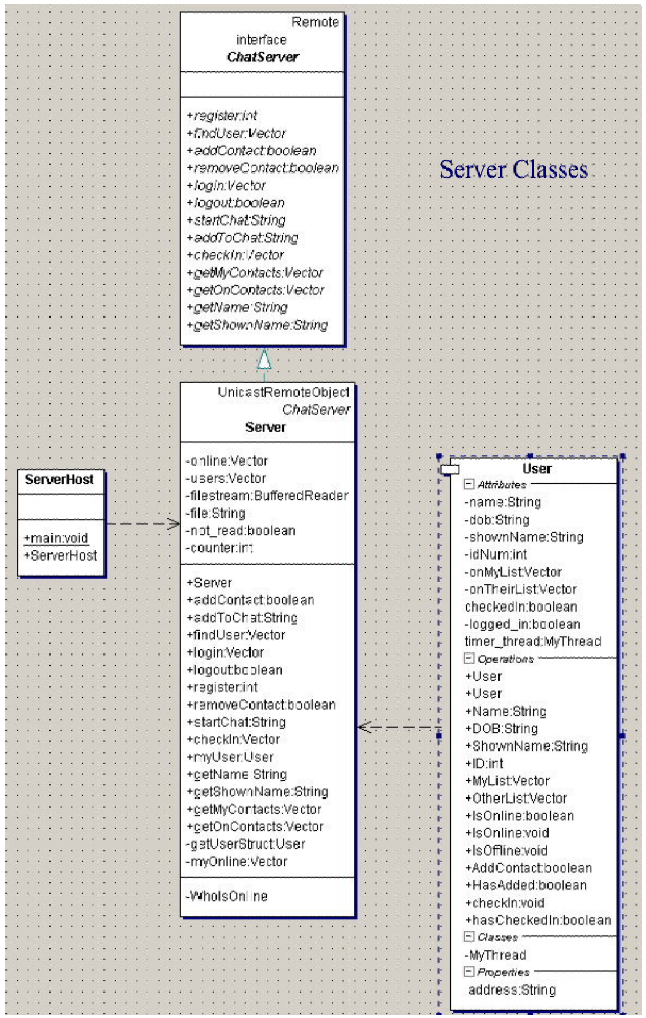
Summary of omissions:

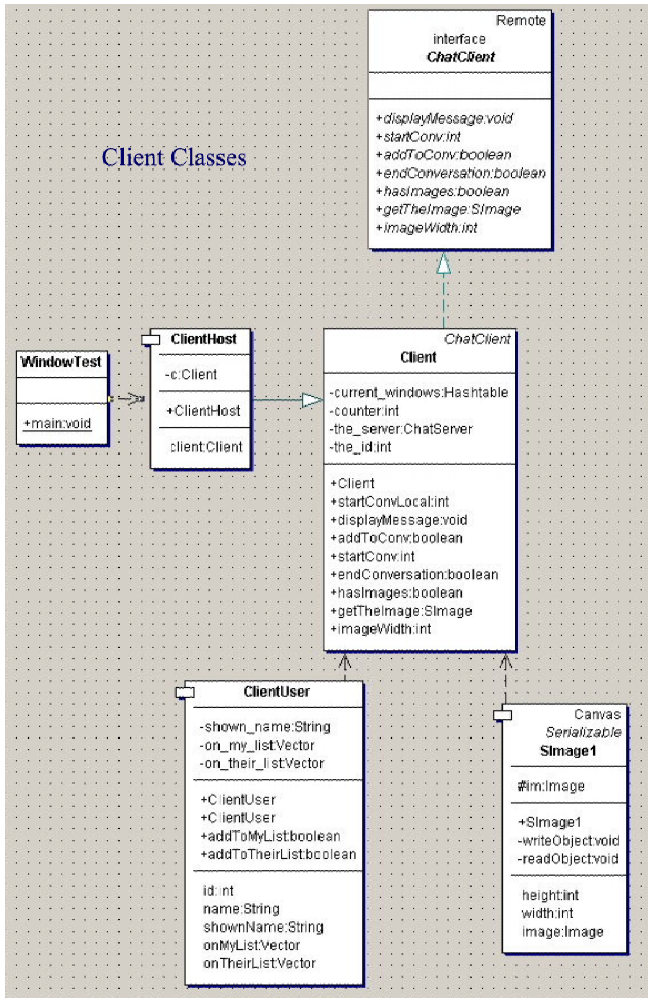
- Some non-critical features omitted
    - No limit on number of conversations.
    - Inability to add another user to an existing conversation.
    - No ability to block another user or to remove a person from a contact list.
    - No implementation to turn off speech or video in a particular window
    - No allowance for emoticons.

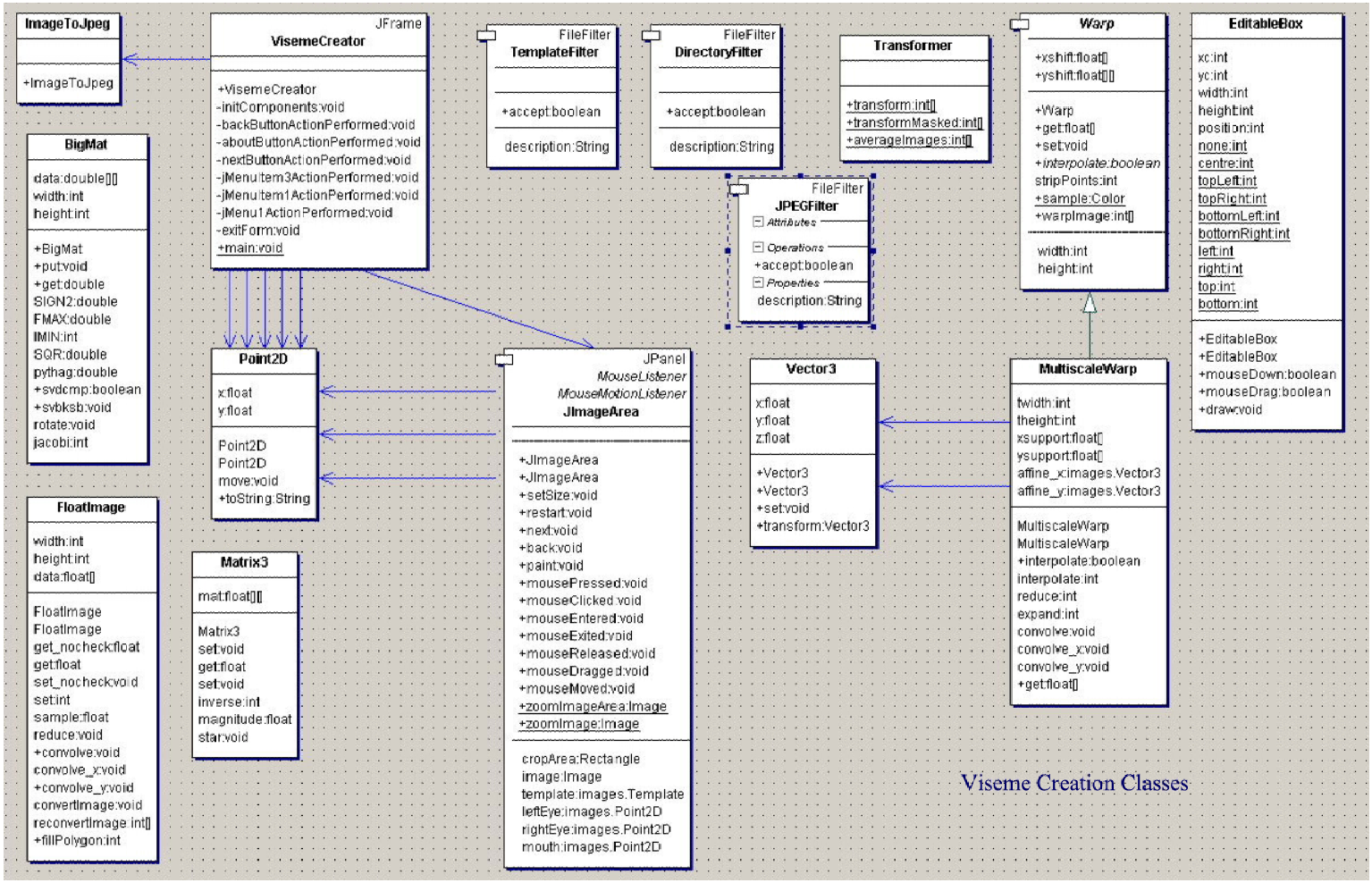
- Alterations to original specifications
    - Not platform independent due to use of MSSAPI. The JSAPI did not have sufficient depth of information available to retrieve data on the current phoneme.
    - Use of Java3D omitted. This simplified the code, made the program run quicker and increased the potential for platform independence, as Java3D has not been implemented on many platforms.

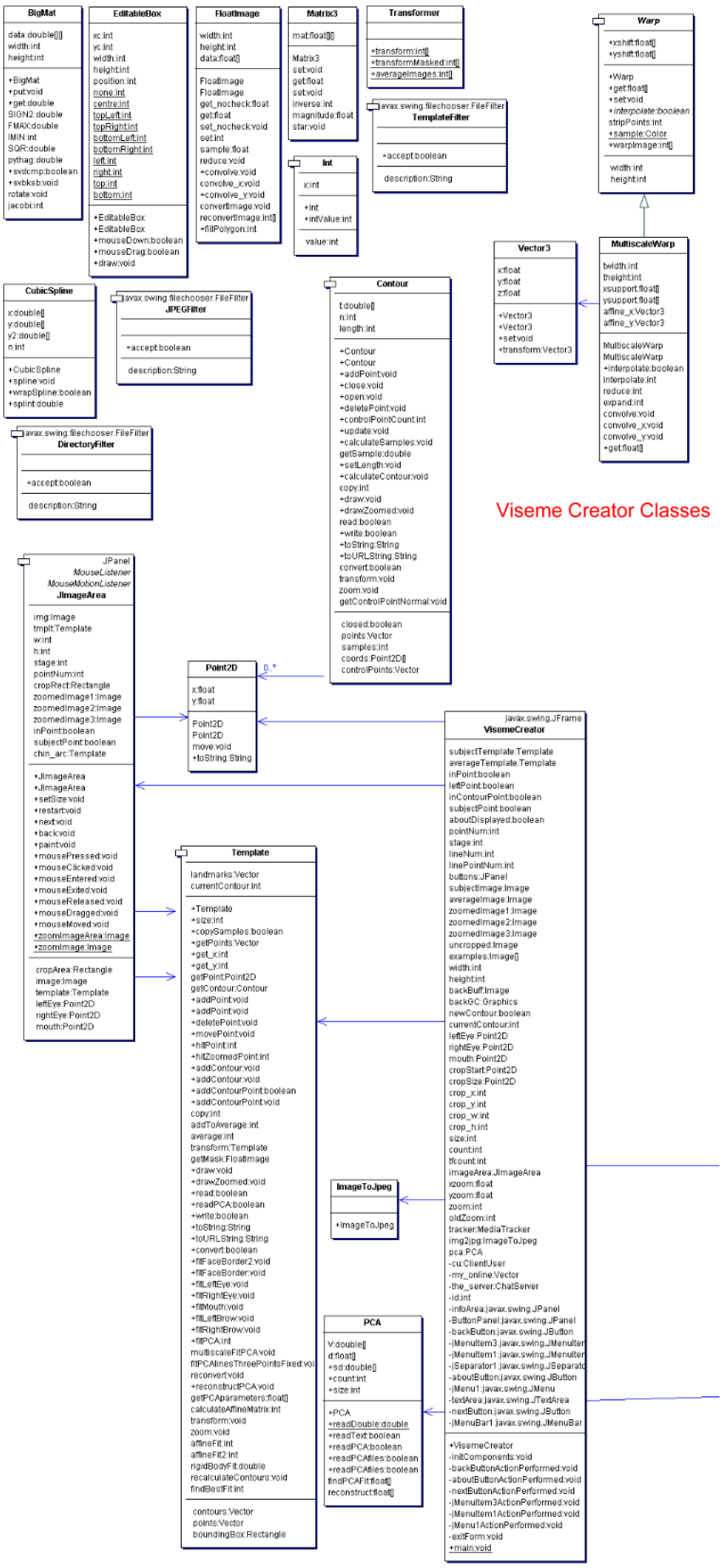
Appendix G - UML

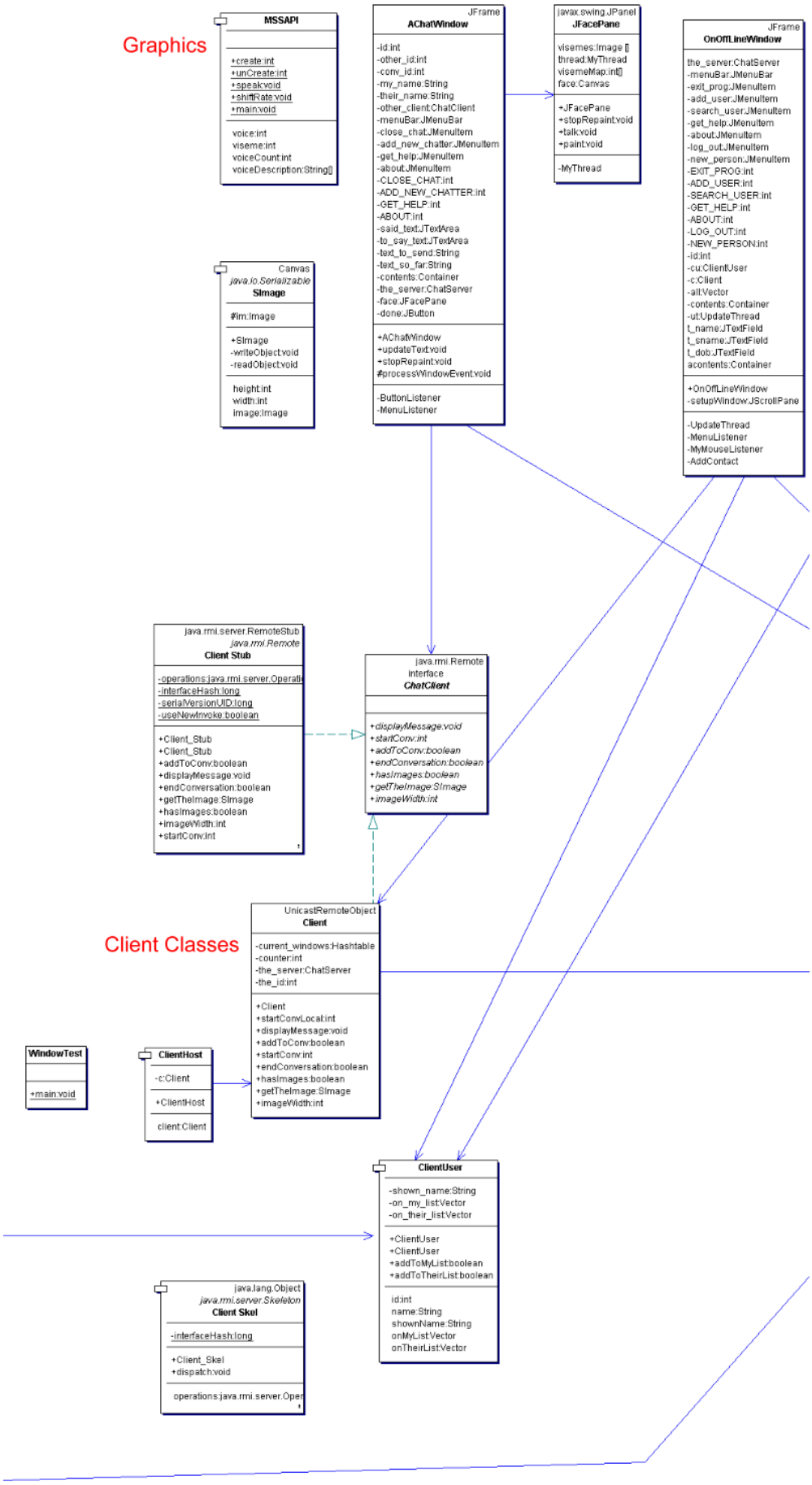


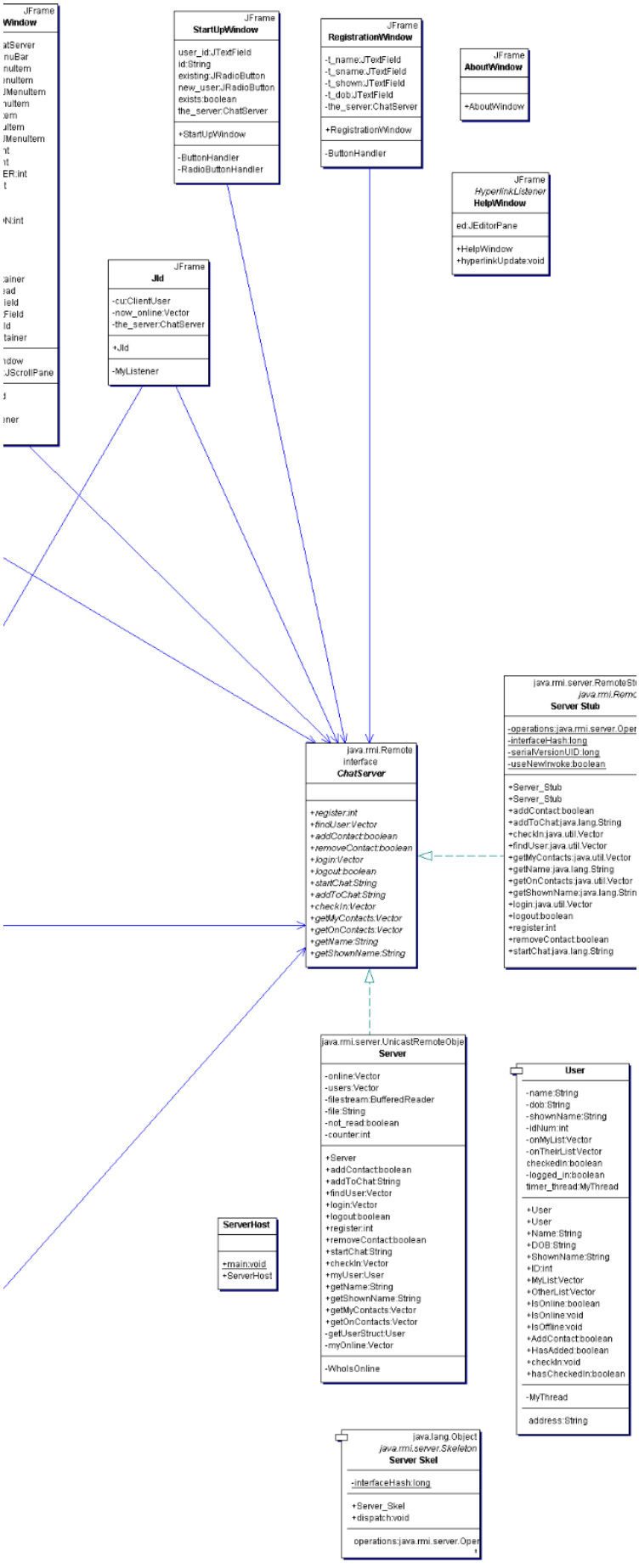

Server Classes
Remote
interface
ChatServer
+register:int
+findUser:Vector
+addContact:boolean
+removeContact:boolean
+login:Vector
+logout:boolean
+startChat:String
+addToChat:String
+checkIn:Vector
+getMyContacts:Vector
+getOnContacts:Vector
+getName:String
+getShownName:String
UnicastRemoteObject
ChatServer
Server
-online:Vector
-users:Vector
-filestream:BufferedReader
-file:String
-not_read:boolean
-counter:int
+Server
+addContact:boolean
+addToChat:String
+findUser:Vector
+login:Vector
+logout:boolean
+register:int
+removeContact:boolean
+startChat:String
+checkIn:Vector
+myUser:User
+getName String
+getShownName:String
+getMyContacts:Vector
+getOnContacts:Vector
-getUserStruct:User
-myOnline:Vector
-WhoIsOnline
ServerHost
+main:void
+ServerHost
User
Attributes
-name:String
-dob:String
-shownName:String
-idNum:int
-onMyList:Vector
-onTheirList:Vector
checkedIn:boolean
-logged_in:boolean
timer_thread:MyThread
Operations
+User
+User
+Name:String
+DOB:String
+ShownName:String
+ID:int
+MyList:Vector
+OtherList:Vector
+IsOnline:boolean
+IsOnline:void
+IsOffline:void
+AddContact:boolean
+HasAdded:boolean
+checkIn:void
+hasCheckedIn:boolean
Classes
-MyThread
Properties
address:String


Client Classes
Remote
interface
ChatClient
+displayMessage:void
+startConv:int
+addToConv:boolean
+endConversation:boolean
+hasImages:boolean
+getTheImage:SImage
+imageWidth:int
WindowTest
+main:void
ClientHost
-c:Client
+ClientHost
client:Client
ChatClient
Client
-current_windows:Hashtable
-counter:int
-the_server:ChatServer
-the_id:int
+Client
+startConvLocal:int
+displayMessage:void
+addToConv:boolean
+startConv:int
+endConversation:boolean
+hasImages:boolean
+getTheImage:SImage
+imageWidth:int
ClientUser
-shown_name:String
-on_my_list:Vector
-on_their_list:Vector
+ClientUser
+ClientUser
+addToMyList:boolean
+addToTheirList:boolean
id:int
name:String
shownName:String
onMyList:Vector
onTheirList:Vector
Canvas
Serializable
SImage1
#im:Image
+SImage1
-writeObject:void
-readObject:void
height:int
width:int
image:Image


ImageToJpeg
+ImageToJpeg
JFrame
VisemeCreator
+VisemeCreator
-initComponents:void
-backButtonActionPerformed:void
-aboutButtonActionPerformed:void
-nextButtonActionPerformed:void
-jMenuItem3ActionPerformed:void
-jMenuItem1ActionPerformed:void
-jMenu1ActionPerformed:void
-exitForm:void
+main:void
FileFilter
TemplateFilter
+accept:boolean
description:String
FileFilter
DirectoryFilter
+accept:boolean
description:String
Transformer
+transform:int[]
+transformMasked:int[]
+averageImages:int[]
FileFilter
JPEGFilter
Attributes
Operations
+accept:boolean
Properties
description:String
Warp
+xshift:float[]
+yshift:float[][]
+Warp
+get:float[]
+set:void
+interpolate:boolean
stripPoints:int
+sample:Color
+warpImage:int[]
width:int
height:int
EditableBox
xc:int
yc:int
width:int
height:int
position:int
none:int
centre:int
topLeft:int
topRight:int
bottomLeft:int
bottomRight:int
left:int
right:int
top:int
bottom:int
+EditableBox
+EditableBox
+mouseDown:boolean
+mouseDrag:boolean
+draw:void
BigMat
data:double[][]
width:int
height:int
+BigMat
+put:void
+get:double
SIGN2:double
FMAX:double
IMIN:int
SQR:double
pythag:double
+svdcmp:boolean
+svbksb:void
rotate:void
jacobi:int
Point2D
x:float
y:float
Point2D
Point2D
move:void
+toString:String
JPanel
MouseListener
MouseMotionListener
JImageArea
+JImageArea
+JImageArea
+setSize:void
+restart:void
+next:void
+back:void
+paint:void
+mousePressed:void
+mouseClicked:void
+mouseEntered:void
+mouseExited:void
+mouseReleased:void
+mouseDragged:void
+mouseMoved:void
+zoomImageArea:Image
+zoomImage:Image
cropArea:Rectangle
image:Image
template:images.Template
leftEye:images.Point2D
rightEye:images.Point2D
mouth:images.Point2D
Vector3
x:float
y:float
z:float
+Vector3
+Vector3
+set:void
+transform:Vector3
MultiscaleWarp
twidth:int
theight:int
xsupport:float[]
ysupport:float[]
affine_x:images.Vector3
affine_y:images.Vector3
MultiscaleWarp
MultiscaleWarp
+interpolate:boolean
interpolate:int
reduce:int
expand:int
convolve:void
convolve_x:void
convolve_y:void
+get:float[]
FloatImage
width:int
height:int
data:float[]
FloatImage
FloatImage
get_nocheck:float
get:float
set_nocheck:void
set:int
sample:float
reduce:void
+convolve:void
convolve_x:void
+convolve_y:void
convertImage:void
reconvertImage:int[]
+fillPolygon:int
Matrix3
mat:float[][]
Matrix3
set:void
get:float
set:void
inverse:int
magnitude:float
star:void
Viseme Creation Classes


BigMat
data:double[][]
width:int
height:int
+BigMat
+put:void
+get:double
SIGN2:double
FMAX:double
IMIN:int
SQR:double
pythag:double
+svdcmp:boolean
+svbksb:void
rotate:void
jacobi:int
EditableBox
xc:int
yc:int
width:int
height:int
position:int
none:int
centre:int
topLeft:int
topRight:int
bottomLeft:int
bottomRight:int
left:int
right:int
top:int
bottom:int
+EditableBox
+EditableBox
+mouseDown:boolean
+mouseDrag:boolean
+draw:void
FloatImage
width:int
height:int
data:float[]
FloatImage
FloatImage
get_nocheck:float
get:float
set_nocheck:void
set:int
sample:float
reduce:void
+convolve:void
convolve_x:void
+convolve_y:void
convertImage:void
reconvertImage:int[]
-fillPolygon:int
Matrix3
mat:float[][]
Matrix3
set:void
get:float
set:void
inverse:int
magnitude:float
star:void
Transformer
+transform:int[]
+transformMasked:int[]
+averageImages:int[]
javax.swing.filechooser.FileFilter
TemplateFilter
+accept:boolean
description:String
Int
x:int
+Int
+intValue:int
value:int
Warp
+xshift:float[]
+yshift:float[]
+Warp
+get:float[]
+set:void
+interpolate:boolean
stripPoints:int
+sample:Color
+warpImage:int[]
width:int
height:int
Vector3
x:float
y:float
z:float
+Vector3
+Vector3
+set:void
+transform:Vector3
MultiscaleWarp
twidth:int
theight:int
xsupport:float[]
ysupport:float[]
affine_x:Vector3
affine_y:Vector3
MultiscaleWarp
MultiscaleWarp
+interpolate:boolean
interpolate:int
reduce:int
expand:int
convolve:void
convolve_x:void
convolve_y:void
+get:float[]
CubicSpline
x:double[]
y:double[]
y2:double[]
n:int
+CubicSpline
+spline:void
+wrapSpline:boolean
+splint:double
javax.swing.filechooser.FileFilter
JPEGFilter
+accept:boolean
description:String
Contour
t:double[]
n:int
length:int
+Contour
+Contour
+addPoint:void
+close:void
+open:void
+deletePoint:void
+controlPointCount:int
+update:void
+calculateSamples:void
getSample:double
+setLength:void
+calculateContour:void
copy:int
+draw:void
+drawZoomed:void
read:boolean
+write:boolean
+toString:String
+toURLString:String
convert:boolean
transform:void
zoom:void
getControlPointNormal:void
closed:boolean
points:Vector
samples:int
coords:Point2D[]
controlPoints:Vector
javax.swing.filechooser.FileFilter
DirectoryFilter
+accept:boolean
description:String
Viseme Creator Classes
JPanel
MouseListener
MouseMotionListener
JImageArea
img:Image
tmplt:Template
w:int
h:int
stage:int
pointNum:int
cropRect:Rectangle
zoomedImage1:Image
zoomedImage2:Image
zoomedImage3:Image
inPoint:boolean
subjectPoint:boolean
chin_arc:Template
+JImageArea
+JImageArea
+setSize:void
+restart:void
+next:void
+back:void
+paint:void
+mousePressed:void
+mouseClicked:void
+mouseEntered:void
+mouseExited:void
+mouseReleased:void
+mouseDragged:void
+mouseMoved:void
+zoomImageArea:Image
+zoomImage:Image
cropArea:Rectangle
image:Image
template:Template
leftEye:Point2D
rightEye:Point2D
mouth:Point2D
Point2D
x:float
y:float
Point2D
Point2D
move:void
+toString:String
0..*
javax.swing.JFrame
VisemeCreator
subjectTemplate:Template
averageTemplate:Template
inPoint:boolean
leftPoint:boolean
inContourPoint:boolean
subjectPoint:boolean
aboutDisplayed:boolean
pointNum:int
stage:int
lineNum:int
linePointNum:int
buttons:JPanel
subjectImage:Image
averageImage:Image
zoomedImage1:Image
zoomedImage2:Image
zoomedImage3:Image
uncropped:Image
examples:Image[]
width:int
height:int
backBuff:Image
backGC:Graphics
newContour:boolean
currentContour:int
leftEye:Point2D
rightEye:Point2D
mouth:Point2D
cropStart:Point2D
cropSize:Point2D
crop_x:int
crop_y:int
crop_w:int
crop_h:int
size:int
count:int
tfcount:int
imageArea:JImageArea
xzoom:float
yzoom:float
zoom:int
oldZoom:int
tracker:MediaTracker
img2jpg:ImageToJpeg
pca:PCA
-cu:ClientUser
-my_online:Vector
-the_server:ChatServer
-id:int
-infoArea:javax.swing.JPanel
-ButtonPanel:javax.swing.JPanel
-backButton:javax.swing.JButton
-jMenuItem3:javax.swing.JMenuItem
-jMenuItem1:javax.swing.JMenuItem
-jSeparator1:javax.swing.JSeparator
-aboutButton:javax.swing.JButton
-jMenu1:javax.swing.JMenu
-textArea:javax.swing.JTextArea
-nextButton:javax.swing.JButton
-jMenuBar1:javax.swing.JMenuBar
+VisemeCreator
-initComponents:void
-backButtonActionPerformed:void
-aboutButtonActionPerformed:void
-nextButtonActionPerformed:void
-jMenuItem3ActionPerformed:void
-jMenuItem1ActionPerformed:void
-jMenu1ActionPerformed:void
-exitForm:void
+main:void
Template
landmarks:Vector
currentContour:int
+Template
+size:int
+copySamples:boolean
+getPoints:Vector
+get_x:int
+get_y:int
getPoint:Point2D
getContour:Contour
+addPoint:void
+addPoint:void
+deletePoint:void
+movePoint:void
+hitPoint:int
+hitZoomedPoint:int
+addContour:void
+addContour:void
+addContourPoint:boolean
+addContourPoint:void
copy:int
addToAverage:int
average:int
transform:Template
getMask:FloatImage
+draw:void
+drawZoomed:void
+read:boolean
+readPCA:boolean
+write:boolean
+toString:String
+toURLString:String
+convert:boolean
+fitFaceBorder2:void
+fitFaceBorder:void
+fitLeftEye:void
+fitRightEye:void
+fitMouth:void
+fitLeftBrow:void
+fitRightBrow:void
+fitPCA:int
multiscaleFitPCA:void
fitPCAlinesThreePointsFixed:void
reconvert:void
+reconstructPCA:void
getPCAparameters:float[]
calculateAffineMatrix:int
transform:void
zoom:void
affineFit:int
affineFit2:int
rigidBodyFit:double
recalculateContours:void
findBestFit:int
contours:Vector
points:Vector
boundingBox:Rectangle
ImageToJpeg
+ImageToJpeg
PCA
V:double[]
d:float[]
+sd:double[]
+count:int
+size:int
+PCA
+readDouble:double
+readText:boolean
+readPCA:boolean
+readPCAfiles:boolean
+readPCAfiles:boolean
findPCAFit:float[]
reconstruct:float[]


Graphics
MSSAPI
+create:int
+unCreate:int
+speak:void
+shiftRate:void
+main:void
voice:int
viseme:int
voiceCount:int
voiceDescription:String[]
JFrame
AChatWindow
-id:int
-other_id:int
-conv_id:int
-my_name:String
-their_name:String
-other_client:ChatClient
-menuBar:JMenuBar
-close_chat:JMenuItem
-add_new_chatter:JMenuItem
-get_help:JMenuItem
-about:JMenuItem
-CLOSE_CHAT:int
-ADD_NEW_CHATTER:int
-GET_HELP:int
-ABOUT:int
-said_text:JTextArea
-to_say_text:JTextArea
-text_to_send:String
-text_so_far:String
-contents:Container
-the_server:ChatServer
-face:JFacePane
-done:JButton
+AChatWindow
+updateText:void
+stopRepaint:void
#processWindowEvent:void
-ButtonListener
-MenuListener
javax.swing.JPanel
JFacePane
visemes:Image []
thread:MyThread
visemeMap:int[]
face:Canvas
+JFacePane
+stopRepaint:void
+talk:void
+paint:void
-MyThread
JFrame
OnOffLineWindow
the_server:ChatServer
-menuBar:JMenuBar
-exit_prog:JMenuItem
-add_user:JMenuItem
-search_user:JMenuItem
-get_help:JMenuItem
-about:JMenuItem
-log_out:JMenuItem
-new_person:JMenuItem
-EXIT_PROG:int
-ADD_USER:int
-SEARCH_USER:int
-GET_HELP:int
-ABOUT:int
-LOG_OUT:int
-NEW_PERSON:int
-id:int
-cu:ClientUser
-c:Client
-alt:Vector
-contents:Container
-ut:UpdateThread
t_name:JTextField
t_sname:JTextField
t_dob:JTextField
acontents:Container
+OnOffLineWindow
-setupWindow:JScrollPane
-UpdateThread
-MenuListener
-MyMouseListener
-AddContact
Canvas
java.io.Serializable
SImage
#im:Image
+SImage
-writeObject:void
-readObject:void
height:int
width:int
image:Image
java.rmi.server.RemoteStub
java.rmi.Remote
Client Stub
-operations:java.rmi.server.Operati
-interfaceHash:long
-serialVersionUID:long
-useNewInvoke:boolean
+Client_Stub
+Client_Stub
+addToConv:boolean
+displayMessage:void
+endConversation:boolean
+getTheImage:SImage
+hasImages:boolean
+imageWidth:int
+startConv:int
java.rmi.Remote
interface
ChatClient
+displayMessage:void
+startConv:int
+addToConv:boolean
+endConversation:boolean
+hasImages:boolean
+getTheImage:SImage
+imageWidth:int
Client Classes
UnicastRemoteObject
Client
-current_windows:Hashtable
-counter:int
-the_server:ChatServer
-the_id:int
+Client
+startConvLocal:int
+displayMessage:void
+addToConv:boolean
+startConv:int
+endConversation:boolean
+hasImages:boolean
+getTheImage:SImage
+imageWidth:int
WindowTest
+main:void
ClientHost
-c:Client
+ClientHost
client:Client
ClientUser
-shown_name:String
-on_my_list:Vector
-on_their_list:Vector
+ClientUser
+ClientUser
+addToMyList:boolean
+addToTheirList:boolean
id:int
name:String
shownName:String
onMyList:Vector
onTheirList:Vector
java.lang.Object
java.rmi.server.Skeleton
Client Skel
-interfaceHash:long
+Client_Skel
+dispatch:void
operations:java.rmi.server.Oper



Server Classes

Appendix H - Maintenance Document

To run the client code, ensure that a network connection is present and then type "chat" in the directory of the program. To run the server code go to the directory of the program and type "java –Djava.security.policy=policy.txt ServerHost". This indicates to the program the location of the current security policy file, which is overwritten by the RMI security manager once the program is started. Ensure that the rmiregistry is also running in the same directory as the server, start this by typing rmiregistry at the command line. If this fails ensure that the rmiregistry is not already running, and that the path is set to include the directory where this program is stored. More information is available in the index.html file provided in the home directory of the program.

If the program needs to be recompiled it is necessary to ensure that both the server and client computers have all the relevant files, especially stubs for both the client and the server. Once the code has been recompiled using javac, the RMI classes need to be dealt with. Typing "rmic Client then rmic Server" in the directory of the program does this.

The system was mainly tested through repeated use, with particular attention being paid to ensure that all possible options had been covered. Of course, this is not possible and there are still some bugs remaining, and probably some to be discovered.

Further testing needs to be done to monitor what happens when either the server or client program fails in some way. Often the program keeps running but with reduced functionality. Ways need to be developed to inform the user when a problem has occurred, and that they should restart their program. Alternately the client should deal with any exceptions thrown more thoroughly than the solutions in the current code. Further investigation needs to be carried out on the system to work out why users are randomly, and very occasionally, removed from the online list in the server when they are still in contact. Some problems were encountered when holding conversations between users with two digit id numbers. This issue has hopefully been fixed but testing was minimal due to time constraints.

Name Emma Russell

First Supervisor Bernard Tiddeman

Second Supervisor Alan Ruddle

Basic Criteria	Comment	Grade
Understanding the Problem	Shows a good understanding of the problem from a software engineering point of view	B
Proper Software Engineering Process	Good plane and process model overall excellent software engineering.	A
Achievement of main objectives	All the main objectives were achieved in full	A
Structure and completeness of report	The report is well structured and complete	A
Structure and completeness of presentation	The presentation was interesting and accessible	A
Additional Criteria		
Knowledge of the Literature	Demonstrates a sound knowledge of the literature. Could have made more use of references	B/C
Critical Evaluation of Previous Work	A good evaluation of related work which addressed the main technologies	B
Critical Evaluation of own work	A thorough evaluation of own work, if anything overly harsh	A
Justification of design decisions	All the main design decisions were justified	B
Solution of any conceptual difficulties	All conceptual difficulties were resolved satisfactorily	B
Achievement in full of all objectives	All the objectives were achieved to my satisfaction	B
Quality of Software	The software worked well, without obvious bugs. A usable user interface was provided.	A
Ambition and Scope of Project	This appeared a very ambitious project. The provision of some software by the supervisor meant it was realistically achievable as a final year project	A
Exceptional Criteria		
Originality of concept, design or analysis	An original and interesting project successfully concluded.	B
Adventure	Yes	B
Inclusion of publishable material	Submitted for publication	B

Recommended Grade: 18
This was an ambitious project, which was successfully completed and well engineered.

	Basic Criteria				
Name	Understanding of the problem	Software Engineering Process & Plan	Achieved main Objectives	Report Quality	Presentation
Emma Russell	A	A	A	A	A

			Additional Criteria		
Software Quality	Understanding of the problem	Knowledge of Literature	Critical evaluation of literature	Critical evaluation of own work	Justification of Design Decisions
A	A	B	B	A	A

Solution of Conceptual Problems	Achieved all objectives	Exceptional Criteria: Evidence of Originality	Exceptional Criteria: Evidence of Adventure	Exceptional Criteria: Inclusion of Publishable Material	Proposed Grade: Total
A	A	B	C	B	19

Comments
Emma worked on this project very independently and taught herself several new pieces of technology. She produced a very nice piece of software and tackled some tricky problems, particularly in the communication aspects of the project (e.g. passing images via RMI). The software has some original aspects and a paper has been submitted to the PGnet conference.

Wireless Speakers
Joint Honours Project

Julian Smith

Submitted: 24th April 2003

Supervised by:
Dr. Graham Kirby

Abstract

The aim of Wireless Speakers was to create an application that allows multiple users to access their own personal music collection from anywhere in the world, using a specially constructed speaker unit. This speaker unit uses a pocket PC with wireless network card to remotely access music files from a server via a wireless network and, if necessary, the Internet.

The project consisted of two main areas – creating a user interface for the pocket PC that allowed users to access their music in a variety of ways, and creating a separate web-based interface designed for a full-size screen that allowed users to build and manage their online libraries.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is 13,018 words long, including project specification and plan.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

Contents Page

Title Page 1

Abstract 2

Declaration 2

Contents 3

Introduction 5

- **Problem Description** 5

Project Details 6

- **Overview** 6
- **The Server** 6
  - Choice of Server 6
  - File Structure 7
- **Cookies** 8
- **Signing Up** 9
  - New User Registration 9
  - Existing Users – Adding a New Device 10
- **The Setup Interface** 10
  - Adding a Track to the Library 10
  - Adding an Album to the Library 11
  - Removing a Track to the Library 11
  - Removing an Album from the Library 11
  - Removing a Playlist from the Library 11
  - Online Help 12
- **The Playback Interface** 12
  - The Homepage and Main Menu 12
  - Windows Media Player Control for pocket Internet Explorer 13
  - Selection and Playback of Music 14
    - Tracks 14
    - Albums 14
    - Existing Playlists 14
    - Random Playlists 14
  - Online Help 15
- **Common Features** 15
  - Creating Playlists 15
  - Changing Default Music 16

- Non-Servlet Classes 16
  - Sorting Classes 16
  - Info Classes 16
  - MultiPartRequest 17
  - Print 17
  - Upload 17
  - User 17
  - Values 18
- **The Speaker Unit 18**

Evaluation and Critical Appraisal 19

- **Standalone Evaluation 19**
- **Known Bugs 20**
  - File Verification 20
  - Playlist Track Removal 20
  - Window Errors 20
  - File Overwriting 21
- **Comparison to Other Systems 21**

Conclusions 23

Appendices 25

- **Appendix A – Problem Definition & Objectives 25**
- **Appendix B – Project Specification and Plan 26**
- **Appendix C – Interim Reports 41**
- **Appendix D – Testing Summary 43**
  - Programming Steps 43
  - System Testing 43
- **Appendix E – Status Report 44**
- **Appendix F – Maintenance Document 45**

Introduction

Problem Description

The aim of this project was to create an application that allowed multiple users to create, maintain and access an online music library through the combination of a web interface and a speaker unit that contained a pocket PC with wireless network capabilities. The web interface would be used to allow users to upload and manage music files whilst the speaker unit would act as a globally portable stereo that could be used anywhere in the world where there was a suitable wireless network.

It was initially intended that the system would include a Java application that could be installed on a pocket PC and a separate web interface to be used as described above. However, early on in the project it transpired that the necessary Java libraries for the Java 2 Micro Edition, namely the Mobile Media API, had not yet been implemented and there was therefore no practical way to achieve this. After alternative methods had been researched it was decided that a web interface would also be used on the pocket PC in connection with the preloaded Windows Media Player. Fortunately, there was a piece of control software available from Microsoft, the Windows Media Player Control for Pocket Internet Explorer, that allows a Media Player object to be embedded in a web page for pocket PC in much the same way as is possible with the standard versions of Internet Explorer. The details of how this was used are laid out in the Project Details section of this report.

As the system has become entirely a web application and the term ‘web interface’ could now apply to any part of the software, I shall refer in the remainder of this report to what was originally planned to be the web interface as the ‘Setup Interface’ and the interface for the pocket PC as the ‘Playback Interface’. These terms are also used in the contents page.

The project has been very successful in achieving its objectives, as can be seen from the Status Report in Appendix E of this report. Every one of the original objectives has been met and the majority of any remaining instabilities in the code are a result of cross-platform issues with the Setup Interface, mostly connected with different browsers' interpretations of JavaScript which is used to ensure that required fields are filled out in the html forms that are used. These are described in the 'Evaluation and Critical Appraisal' section of this report along with a few other known bugs. Despite these minor issues, the system has all of the functionality that it was intended to have and can, I think be fairly deemed to be successful.

Project Details

Overview

The two interfaces that make up the bulk of the project divide the system into two main sections. These interfaces are constructed almost entirely from Java servlets with the exception of some static html pages such as the main menu of the Playback Interface, and some Java Server Pages such as the online help for the Setup Interface. Some of these servlets are used by both interfaces, but most can only be accessed from one or the other. Underneath these servlet classes are ten non-servlet classes that mostly deal with the IO to and from media files and meta-data files in the users' libraries.

The system is now entirely web-based, as it was not possible to develop an application for the iPAQ that could handle audio playback. This has meant that some of the code could be centralised and, as mentioned above, used for both interfaces. Both interfaces were designed to be easy to navigate, with only the information on each page that was needed. The Playback Interface is more graphical to make it more interesting to use and to get the best out of the smaller screen of the iPAQ. Text links can be reasonably difficult to see and use on any pocket PC so these have been kept to a minimum and .gif files have been used extensively as link anchors in their place.

The Setup Interface is more functional than the Playback Interface and so is much simpler in both its layout and style. This fits its purpose as it is only used in order to make the Playback Interface useable by having a library of music to choose from.

The Server

Choice of Server

As this project has become purely a web application, an appropriate web server that could handle the application's requirements was essential. The server used for this project was the Apache Tomcat 4.0 server, available free from http://jakarta.apache.org and capable of handling Java Servlets and Java Server Pages in addition to meeting standard web server requirements.

This has proved to be very reliable during the course of the project and has fulfilled everything needed of it. It was reasonably complex to set up though and I am grateful to Google.com for finding me some invaluable tutorials on its configuration and use. It has been particularly useful during testing as it provides web pages detailing any runtime errors which would not be identifiable otherwise as the code has to be tested in a servlet context and not just a standard Java Runtime Environment.

This is not, however, a suitable server for a larger-scale project so research would need to be carried out into a more suitable option if one wanted to adapt and expand this project to, for example, a commercial purpose.

File Structure

There are four directories in the ROOT folder that are part of the initial application. These are WEB-INF, setup, help and common which contain respectively the class files for all of the Java code, the menu, welcome and help pages for the Setup Interface, the help files for the Playback Interface, and the initial default music file 'welcome.wma'. There are also the files users.txt, index.htm and menu.htm and 27 image files for the Playback Interface.

Each user has a home directory in the ROOT directory (named with their username), which contains by default the files 'tracks.wsl', 'default.asx' and 'albums.txt', and the sub-directories 'playlists' and 'Unknown'. 'Tracks.wsl' is a Wireless Speakers Library file which holds a line of meta-data for each track in a user's library in the following format:

"Title""Album""Track Number""Artist""Genre""File Extension"

'Default.asx' is a Windows playlist file that is set to contain the user's current default music and is accessed by the Playback Interface homepage. 'Albums.txt' is a plain text file that holds a line of meta-data for each album in the user's library in the following format:

"Title""Artist""Genre"

Each new artist in the user's library is given a sub-directory in the user's home directory with any albums having a sub-directory within this that includes the media files for the album and a file 'this.asx' that is a playlist file referencing the album's tracks in the correct order. All individual tracks are stored in a sub-directory 'Unknown' in their artist's directory or the top-level 'Unknown' directory if no artist is specified. All playlists that a user creates, including the most recent random playlist, are stored as '.asx' files in the user's 'playlists' directory. The diagram below shows an example of the ROOT directory with some sub-directories.

Diagram Showing Example File Structure

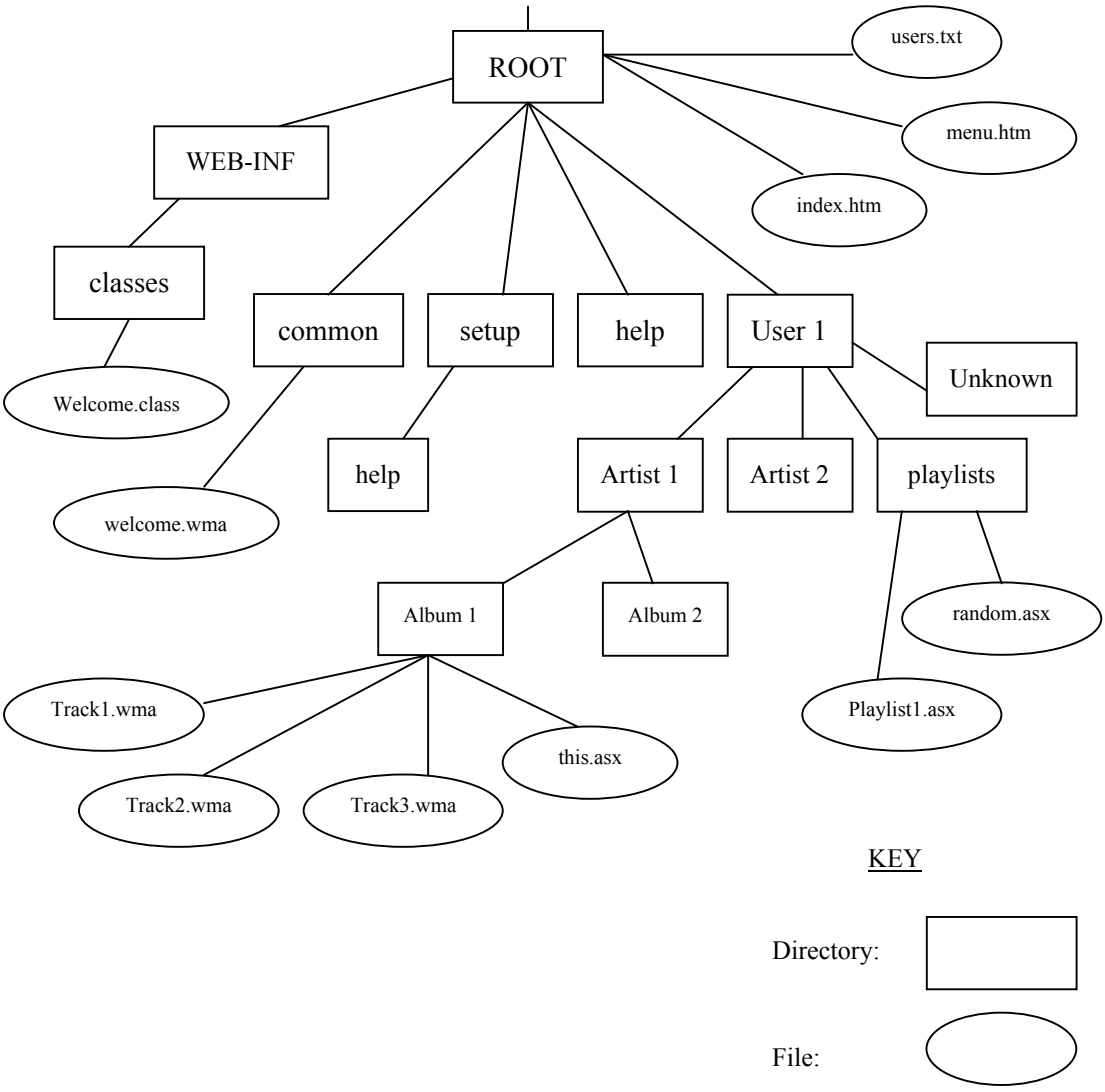


Fig 1.0 – Example file structure

Cookies

Wireless Speakers uses cookies to identify users, of which there are several different types used according to what kind of device is accessing the system. A registered speaker unit will have been sent a cookie with the name 'username' and the appropriate username as its value. A desktop or laptop machine that is being used for access to the Setup Interface will have been sent a cookie with the name 'setup' and a username as its value. The section on 'Signing Up' below describes how the correct type of cookie is sent to new users or devices being added by existing users.

Signing Up

There is a single servlet, called 'Welcome.java', that handles the initial stage of the signup procedure. This is the same servlet that provides known devices with the appropriate homepage of their users. If a device is not recognised, it provides two choices: to add the current device to an existing user's profile or to register as a new user. 'Welcome.java' and the two servlets outlined below are used whether the client machine is to be used for accessing the Setup or Playback Interfaces. All pages contain the question mark graphic that links to the relevant page in the online help. As either a desktop or pocket device could be in use, the Playback Interface help is used for ease of viewing.

New User Registration

The servlet 'AddUser.java' handles the procedure for registering new users. It is accessed by selecting the 'New User' link from the welcome page. Initially, there is a textbox in which to enter the desired username and two checkboxes for the user to identify firstly whether the current device is a mobile speaker unit and secondly whether it is a shared computer. The isUser() and checkFile() methods in User.java are used to check whether the desired username is already in use, or is the same as one of the system filenames respectively. If either is the case, the user is informed and a textbox is provided for an alternative username. The servlet remembers the checkbox values and displays them to the user on the same page.

If the username is not in use, the create() method of User.java is called which does the following: creates a directory in the ROOT directory with the username as its name; creates sub-directories in that directory called 'playlists' and 'Unknown' (the root folder for tracks of unknown artists); adds the new username to 'users.txt' (a plain text file that contains a list of current usernames delimited by new lines); creates the empty files 'tracks.wsl' and 'albums.txt' in the user's home directory; sets the new user's default choice of music to 'welcome.wma' which is stored in the 'common' directory.

The value of the speaker unit checkbox determines the name of the cookie that is sent to the client machine. If the box is checked, the cookie name is set to 'username' and that machine will be automatically guided to the Playback Interface whenever it accesses Wireless Speakers in the future, otherwise the cookie name is set to 'setup' and the machine will be guided to the Setup Interface in the future.

If the 'shared computer' box is checked then the cookie length is left on its default which means it will expire when the user quits their web browser and they or anyone else who accesses Wireless Speakers through that machine in the future will have to go through the signup procedure again. If the box is not checked, a persistent cookie is sent with a lifespan of one year and the system will recognise the client for the duration of that year.

Existing Users - Adding a Device

The servlet 'AddDevice.java' handles adding a device for an existing user. Initially there is a textbox for the appropriate username to be entered and the same checkboxes as described above in 'New User Registration'. The isUser() method of User.java is called to check if this is a valid username. If it is then an appropriate cookie is sent to the client and a page is returned reporting the successful adding of the client device along with a link to the Welcome servlet that will provide the correct homepage for the user. If it is not valid then the user is informed and another textbox is provided for re-submission.

The Setup Interface

The Setup Interface handles all of the facilities for adding to a library, deleting from it, and making other changes. If a 'setup' cookie (see ‘Cookies’ above) is sent by a client then the Setup Interface index is returned by the Welcome servlet. This is a framed page that contains a menubar on the left of the page that is written in plain html and a welcome page as the main frame that describes a bit about the facilities available from Wireless Speakers and provides the same links as the menubar.

Adding a Track to the Library

This action consists of three pages, all generated by the servlet 'AddTrack.java'. The first page contains a form for submitting the artist and genre of the new track. The getArtists() method of User.java is called to provide a drop-down list of the artists for whom the user has previously uploaded tracks or albums. This field is ignored if anything except white space is entered in the 'New Artist' field.

The 'Continue to File Selection' button submits this data to the servlet, which then returns a page containing a form with hidden values for the data already entered, and a file upload slot and 'Browse' button. The script_validate() method of Print.java is used to generate JavaScript that ensures that a value is entered in the file field. However, this does not check that this is indeed a file and a correct file type. There is, at present, no error checking on submitted files so any kind of file can be submitted and the system will only react if and when it tries to play an incorrect file. In this scenario, the Windows Media Player Control will return an error message saying that the file cannot be played.

On submission of this form, an instance of MultiPartRequest.java is created which saves the submitted file to the specified directory and provides its filename. The filename (without its file extension) is then used as the title for the track and all of the data that has been entered for the track is displayed to the user along with links to add another track or return to the welcome page.

Adding an Album to the Library

This action is very similar to adding tracks, but with some obvious differences. It is handled by the servlet 'AddAlbum.java' but uses the same basic structure. The first page provides form fields for the album's title, the artist (existing or new), the genre and the number of tracks. The album title field is checked to be non-null by the script_validate() method in Print.java and the servlet also returns the same page if the value is just white space.

Assuming correct submission, the servlet then returns a form page with a file slot for each track (according to the number entered on the previous page). The script_validate() method is used to ensure that there is a value for every file slot but the same limitations apply in this regard as for adding tracks. On submission of this form, the servlet creates an instance of MultiPartRequest.java which saves all of the files in the specified directory and returns a page giving all of the submitted details for the album, including a list of the track titles.

Removing a Track from the Library

Removing a track from a user's library is only possible if they have uploaded that track individually. These tracks are identifiable by the fact that they are the only tracks in the library with the album value 'Unknown'. The servlet calls the getTracks() method of User.java and iterates through the list, adding any suitable files to a new list. If there are no individually submitted tracks then the servlet returns an error message and the user can go no further. If there are tracks available to remove, the servlet sets up a form and lists them in a select field. If the form is submitted with no track selected, the same page is returned. If a track is selected, the system attempts to remove it by deleting the media file and then re-writing the user's 'tracks.wsl' file without this track listed in it. User's are prevented from deleting their default music by checking any track submitted with the isDefault() method in User.java. The list of tracks can also be sorted by title, album, artist or genre using the 'TrackSort' class which is described below in the 'Common Features' section.

Removing an Album from the Library

Removing an album is essentially the same as removing a track - it is only possible if there are albums in the user's library; if albums are available they are displayed in the same format in a form page; and a user is prevented from removing their default music from their library. In addition the 'RemoveAlbum' servlet, checks each track in the album with User.isDefault() and prevents the user from deleting the album if one of its tracks is set as their default. The album list can be sorted using the 'AlbumSort' class.

Removing a Playlist from the Library

Removing a playlist is fundamentally different as it does involve deleting any media files. The same

error-checking applies to ensure that a user has at least one playlist that they can delete before being allowed to proceed. There are no sorting facilities available as the playlists are naturally returned in alphabetical order by title and there are no other parameters to sort them by. This is a two-stage process that involves firstly selecting a playlist to remove and then (hopefully) being shown confirmation that the playlist has been deleted. A playlist that is set as the user's default music cannot be removed.

Online Help

The online help for the Setup Interface is a framed page with a menubar on the left and a single Java Server Page (JSP) for the main frame that provides help on specific topics when passed a parameter or an introduction page if no parameter is passed. The online help is always displayed in a new window so that a user will not lose their position in their current action. This window is fixed for presentation to be 800x500 pixels and devoid of its toolbars and menus using the script_help() method in 'Print.java'.

The Playback Interface

The Playback Interface is a web interface that is designed to fit on the screen of a pocket PC. Its features are all geared towards giving the user maximum access to their music in as straightforward a way as possible. It has more visual design than the Setup Interface as it is, realistically, the more fun side of the application and getting the correct music on demand is the main function of the project as a whole.

The Homepage and Main Menu

To meet one of the core objectives of the project, the homepage of the Playback Interface automatically loads and plays the user's default choice of music, which is stored in the playlist file 'default.asx'. It contains a heading welcoming the user, a Media Player object embedded in the centre of the page (see below), and a link to the main menu.

The main menu is a static html page consisting of a background image that occupies most of the screen, and contains the names of different features separated by dashed lines. It is overlaid with an image map to delimit different areas and provide links to different features. It includes a link to the index of the online help for the Playback Interface. Figure 2.0 (below) shows the background image that is used.

Main Menu Display

Fig. 2.0

Windows Media Player Control for Pocket Internet Explorer

Every page that actually plays music back through the speaker unit includes an embedded Media Player object (the Player object) that gives the user some playback controls. The Windows Media Player Control software is needed in order to produce this as embedding media objects is not available as standard in Pocket Internet Explorer.

The panel that is displayed contains the following controls: 'Play' button (becomes 'Pause' when media is playing), 'Stop' button, 'Skip Backwards' and 'Skip Forwards' buttons, and a sliding volume control. There is also a horizontal slider and a counter giving the position in the current track visually and in digits. The Player object does not allow for fast searching through tracks as the data is automatically streamed from its source. There is also no facility for repeat play. Figure 2.1 is a screenshot of the panel that is displayed. It was captured whilst media was playing and so the 'Play' button has been replaced by 'Pause'.

Screenshot of the Media Player Control Panel

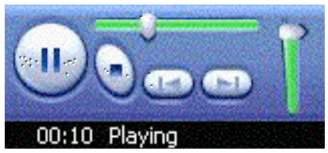


Fig. 2.1

During development there was a recurring problem in that only a very few files that were tested were actually being loaded and played correctly by the Player object. It was discovered that an automatic copyright setting was preventing most of the test files from being shared with another device and the removal of this setting has partly cured the problem. However, mp3 files – the most popular form of

compressed digital media storage – are still not loading properly. This has been a confusing problem as the control software now works perfectly with Windows Media files and is also licensed to use mp3 technology.

Selection and Playback of Music

As can be seen from the main menu graphic (Fig. 2.0), there are three options for selecting music to play – track, album or playlist. The basic method for selecting some music to play is essentially the same for all three. On following the appropriate link, for example 'Choose Track', the user is presented with a list of all the music available or an error message if there is none available. This list is displayed as an html form with a select field listing the available media and a submit button labelled 'Play'.

This form is submitted to the servlet 'Play.java', which sets up a page containing information on the music being played, a Player object (see above) with the appropriate filename as a parameter, and links to the user's homepage and the main menu. Throughout these pages, there is always a link to the relevant page in the online help in case of any confusion.

Tracks

Assuming the user has some music in their library, their tracks are read in from 'tracks.wsl' and displayed in the select field in the order that they were originally uploaded. There are four links in a row at the foot of the page that are labelled 'Title', 'Album', 'Artist' and 'Genre'. Clicking these links will display the same page again but the tracks will have been reordered alphabetically using 'TrackSort.java' (see below) and the chosen parameter. This parameter will also be displayed first on each line, followed by some other identifying information, so that the user can find the value they are looking for.

Albums

Selecting albums is done in a very similar way – the user is provided with a list of their albums that is initially in the order that they were originally uploaded. They can be reordered in the same way as tracks but using 'AlbumSort.java' and by the parameters 'Album Title', 'Artist' and 'Genre'.

Existing Playlists

Playlists that already exist in the user's library can be selected in the same way as tracks and album by following the 'Playlist Options' and 'Choose Playlist' links from the main menu. They are displayed the same way however there are no sorting methods to run as they are automatically returned in alphabetical order and there are no other parameters to sort them by.

Random Playlists

Random playlists can be requested by selecting an artist or genre and the number of tracks desired. The initial values on the selection page are 'All Artists', genre 'Not Selected', and a size of 10. If these

values are left unchanged then the system will generate a random playlist of ten tracks from all the tracks in the library. The artist and genre fields on the form are drop-down lists that include all of the artists and genres that are currently in the user's library. As the genre field is initially set to 'Not Selected', the artist field is used from the submitted form by default. If a user selects a genre from the list then this will override any choice of artist that they have made.

If there are not enough tracks in the library to fulfil the user's request then all of the suitable tracks are returned in a random order. The size of the playlist is always displayed on the final playing page so the user will know if this has happened.

Online Help

The online help is different to the help for the Setup Interface as it needs to be suitable for the small screen size of the pocket PC and it covers mostly different features. There is a framed page that can only be accessed from the link on the main menu, which displays links to the main help topics and back to the main menu. The help information is in a single .htm file with anchors at each topic so the links from specific pages, for example the page for selecting random playlists, go straight to the relevant part of the page and do not bother with the top menu frame.

There is always a link back to the appropriate feature in the interface so that users do not get stuck in the help page, as well as one that goes to the top of the help page itself where there is a full menu of topics to choose from. The help information is as brief as it can be whilst still being reasonably comprehensive, so it is easier to read on the small screen and generally not too long-winded

Common Features

The features outlined below are available through either interface and are handled by the same servlets regardless of which interface is in use. The getCookieInfo() method of User.java is called at the top of each servlet to retrieve the type of cookie sent and the username value.

Creating a Playlist from Existing Tracks

Users can create playlists of any length greater than zero through the servlet 'AddPlaylist.java'. This servlet first provides a form asking for a title for the new playlist. The user will not be able to proceed if no title is entered, or it is all white space. Otherwise the getTracks() method of 'User.java' is called to provide a LinkedList of all the media files in the user's library. A form is then returned with a select box containing all of the tracks and two submit buttons - one to add a track and have the option of adding more, and the other to add a track and create the finished list.

Due to implementation difficulties, there is no facility on this page to sort the tracks by title, album, artist, or genre. The first track in the list is selected by default to ensure that a non-null value is

submitted and the script_validate() method is used to provide JavaScript to back this up in case of an error on the page. If the user does not have any music in their library to choose from, the servlet will return an error message and there will be no way for the user to proceed.

Changing the Default Choice of Music

The process for changing the default music is slightly different between the two interfaces because of the limited size of the iPAQ's screen. In the Playback Interface, the first screen that the user sees gives them the option of choosing a track, album or playlist, and then handles these separately. The pages it returns are identical to the pages returned for selecting tracks, albums or playlists to be played through the speaker unit except that the submit button on the form is labelled 'Select' instead of 'Play'. The first option in the list is selected by default to prevent a null value being submitted, and a confirmation page is returned once the user has selected a new default choice.

In the Setup Interface, the larger screen allows for all three types of media to be displayed at once, cutting out one step. The page contains a form for each type of media that is available in the library so three in total if tracks, albums and playlists are all available. The first option in each list is selected by default and there is a separate submit button for each form. Clicking on one of the buttons submits only the form that it is part of so the system can tell what type of media has been selected. The servlet then returns a confirmation page containing the details of the music that has been selected.

Non-Servlet Classes

Sorting Classes

The classes 'TrackSort' and 'AlbumSort' use the same process to return sorted versions of the lists of tracks and albums that they are passed, using the specified key. They construct a Comparator object based on the value of the key that they are constructed with and then use the Collections.sort() method with the list and the Comparator to produce an ordered version. The keys that can be used with TrackSort are 'title', 'album', 'artist' and 'genre'. The AlbumSort class uses all of these except the 'album' value as 'title' refers to the album's title.

Info Classes

The classes 'TrackInfo' and 'AlbumInfo' are used to store meta-data about tracks and albums that are being manipulated by the system. As all of the variables in them are initialised as 'null', only the parts that are required for a particular function need to be set. This is useful when dealing with servlets as an instance of one of these classes cannot be sent from one servlet to another. However, only the variables that are required need to be sent as String parameters to the receiving servlet instead of extra data or null values.

MultiPartRequest

'MultiPartRequest' is adapted from an example class that is provided in the O'Reilly book 'Java Servlet Programming' by Hunter & Crawford. This class retrieves the file data from the ServletRequest that receives any files submitted through the Setup Interface, saving them to a directory that must be specified in the constructor. It is partly for this reason that the process for uploading tracks and albums is split onto two pages as the artist value and album title for each submitted file is used in creating the directory that they will be stored in. This information is therefore required before the files can be saved and so is submitted on the page prior to file uploading in the interface.

Print

The 'Print' class contains several methods for printing different types of html and JavaScript. Each method takes the calling servlet's PrintWriter as a parameter and many take additional parameters in order to print specific information.

Upload

This class contains two methods, 'track()' and 'album()', which create the meta-data for tracks or albums that is required to search for and play them in the future. The track() method checks if the submitted artist already has a sub-directory in the user's home directory, creating one if it does not exist already. It then creates a FileOutputStream for the user's 'tracks.wsl' file and appends a new line containing data on the new track.

The album() makes the same check for the artist's directory, then creates a sub-directory within it named with the album title and a playlist file called "this.asx" which references all of the album tracks in order. It is this file which is passed to the Media Player object in the Playback Interface if the album is selected to be played. Lastly, the method creates a FileOutputStream for the user's 'albums.txt' file and appends a new line containing data on the new album.

User

The 'User' class contains methods that carry out various actions specific to an individual user or the list of users, which are used throughout the application by different servlet classes. These methods are isUser(), create(), getCookieValue(), getCookieInfo(), getTracks(), getAlbums(), getPlaylists(), getArtists(), isArtist(), checkFile(), getDefault(), isDefault(), setDefault(), and writeAsxFile(). Most of these methods are self-explanatory by their names with the following exceptions: getCookieValue() is used by servlets that are only part of one of the interfaces, whereas getCookieInfo() also returns the type of cookie and thus identifies the interface in use; checkFile() checks whether or not a file in the user's home directory is a system file; writeAsxFile() takes a File object, username and LinkedList of entries and writes to the specified file, creating a Windows playlist file that references the entries in the LinkedList.

Values

The 'Values' class contains final values such as the name of the server, the names of the different cookie types, the maximum track/album size allowed, and so on. If the application were to be distributed then some of these values, such as the server name or path from the servlet classes to the ROOT directory, would need to be generated dynamically.

The Speaker Unit

The speaker unit was constructed by Brian McAndie, one of the department's technicians. It consists of a hi-fi speaker cabinet with a hole cut in the back panel to provide access to the iPAQ's screen. Inside the cabinet, the main speaker cone has been removed and a PC speaker has been put in its place. This has a built-in amplifier and can therefore be connected directly to the headphone socket of the iPAQ, removing the need for a separate amplifier. There is a smaller speaker cone at the top of the cabinet that has been wired to the extra speaker socket on the PC speaker so that both sides of stereo sound are played. The iPAQ with its expansion pack and wireless network card are fixed to the inside of the back panel with the whole of the front face visible including the on/off switch.

Evaluation and Critical Appraisal

Standalone Evaluation

Overall the project has been very successful, achieving all of its broad aims with only some minor requirements not implemented. The Status Report (Appendix E) shows a table of the original objectives as defined in the Problem Definition (Appendix A) against their status at the end of development. Most have been fully implemented and there are only minor features missing, some of which are due to the limitations of the technology used. For example, searching through tracks as they are playing is not possible as the files are streamed and the Media Player Control object used in the Playback Interface does not have the facility to jump to a different point and request the appropriate packets to be sent.

The method of user recognition works well, and although it relies on the use of cookies, this allows users to use other computers to access the interfaces temporarily and either make changes or access their music without fear of leaving their library open to other people. For example, if a user were to take their speaker unit to a friend's house for the evening and wanted to use a feature in the Setup Interface while they were there, they could sign up their friend's computer to their own user profile as a shared computer, carry out the actions that they want, and then leave no trace of having accessed their account once they have closed the Internet browser.

In the Setup Interface, the procedures for adding and removing media are clearly laid out and easy to use. There is the problem (see Known Bugs) of a lack of file verification when adding tracks and albums to the library, but assuming that the user is trying to upload legitimate media files, the system is stable and runs very quickly. The pages only slow down if a large amount of data is being uploaded over a slow network connection.

The interface is designed to be mostly self-explanatory, which is made easier by the limited functionality that is required of it. The online help explains any details that are not obvious from the interface pages themselves.

The Playback Interface is also easy to use and very stable. The only errors that occur come from unplayable media files or other files that are treated as if they were media files. This produces a message from the Media Player object reporting that the specified file is unplayable, however the interface page is still displayed correctly and the system does not fail in any way.

The interface is designed to be easy to navigate, with the minimum of items per page and link graphics that are reasonably large and thus easier to read and select with the stylus. At the same time it uses as few steps as possible to get to the user's choice of action and gives more common actions, such as choosing an album, priority with a link from the main menu instead of placing them in sub-menus.

There is one obvious drawback to the Playback Interface being a web interface and not a Java application, which is that a user will not be able to access anything at all when the speaker unit is not in a wireless environment. Had this interface been built as an application, a user would still be able to run the application in an 'offline' mode when not connected to a network and get access to features such as the online help. The speaker unit now relies entirely on having access to the server before it can provide any functionality at all. This was not, of course, a design decision but a necessary alternative after it emerged that the Mobile Media API, which should have provided libraries for audio playback, has not yet been implemented.

Known Bugs

File Verification

There is no verification built in to the file uploading procedures. In other words it is possible to upload files that are not suitable media files. It is also possible to 'successfully' submit values in the file fields of the Setup Interface forms that are not actually files at all. If unsuitable files are uploaded through the Setup Interface then these will be successfully stored into the user's library and the only problems will arise when the user attempts to play one of these files. The Windows Media Player object that is embedded in the 'Play' page will give an error message stating that the file could not be played because it is missing or unsuitable, however the application will remain stable.

If a value is entered that is not a file then the application will attempt to treat it as a file. The MultiPartRequest class will create an empty file in the specified directory that is named with the value that was entered, however if this value is not in the format 'filename.extension', the system will fail to tokenise the filename correctly and cause an exception. The exception is reported by Tomcat and although the interface will continue to function properly, this is still not an ideal situation.

Playlist Track Removal

Although checks are in place to prevent users from removing their default music, it is still possible to remove a track that is part of a playlist. Fortunately, this does not cause an error when the playlist is selected as the Media Player object just skips the missing file and goes on to the next one. If all of the tracks in the playlist have been removed then it displays a message stating that the playlist file could not be found or opened.

Window Errors

Occasionally the links on the menubar of the Setup Interface open new browser windows instead of opening a page in the main frame. This only occurs when an action is being carried out or a confirmation page is being displayed, but I have not been able to identify the specific circumstance that

causes it and thus prevent it from happening. Refreshing the original window stops this from happening once it has started and so it is presumably part of the servlet code that is at fault.

File Overwriting

There is no check to stop a user adding tracks, albums or playlists that already exist in their library. If adding a track or album, the system will overwrite media files in the library with any that have the same name, artist and album values as the existing ones. The 'tracks.wsl' and 'albums.txt' files are also updated and thus contain the meta-data for any such media twice. However it is worth noting that whilst this will cause some duplication in the meta-data, there will never be files missing that are said to be in the library and the user will effectively just have two options that lead to the same music. This is maintained if the music in question is removed from the library as all meta-data, including any duplicate copies, will be picked out and removed by the system.

Comparison To Other Systems

The defining feature that sets Wireless Speakers apart from the systems discussed in the Context Survey of my Project Specification is the range over which it can be used. It is the only system that provides a music-playing device that is portable beyond about 100 feet from its server, and of course the Wireless Speakers speaker unit is globally portable. There are other advantages that are discussed in reference to each of the alternative systems.

Turtle Beach's Sonic Link product includes a transceiver that attaches to your PC via a parallel port and the line out socket, and a receiver that connects to your home stereo. There is a 2.4GHz remote control that allows you to control the media player on your PC remotely, although with a limited range, and the transceiver then takes the line out signal from your PC and sends it as an analogue radio signal to the receiver on your stereo.

The Sonic Link does not provide any kind of display by which users can choose music to play. It also requires connection to a stereo meaning that it is not as simple to move about as a Wireless Speaker. The range of the signal transmission is about 100 feet so this is effectively only useable within and around the building where the server is kept.

IRemote is an application that enables a pocket PC to act as a remote control for a PC media player, using a wireless network to communicate. Although this is a very different piece of software to Wireless Speakers, from the user's point of view it provides the same functionality as the first half of the Playback Interface in that it is a way of using a pocket PC to remotely select music from a personal library. Having said that, it is fundamentally different in that the music is played back through the server and additional equipment would be required to transmit music from the server and play it somewhere else. Wireless Speakers provides a more complete solution to remotely accessing music as the music is played back at the point of access, wherever that may be.

Thirdly there was the Wireless Sound Power Stereo Speaker Jack System. This consists of a radio transmitter and matching receiver that connect to the line out socket of a PC and a speaker or pair of speakers. The transmitter broadcasts the line out signal at the unlicensed frequency of 2.4GHz. The receiver then plays the music out through the speaker(s) from a maximum range of 100 feet. This is very similar to the Sonic Link system except it has a built-in amplifier and so only requires connection to speakers as opposed to a stereo or set of powered speakers.

It comes with a remote control allowing the user control the media player on their PC from a distance, however its range is still very limited next to the potential range of Wireless Speakers. With both this product and Sonic Link, there is also the potential for interference that would reduce sound quality as they both broadcast an analogue radio signal whereas all data transmission with Wireless Speakers is digitally transmitted and is checked for its integrity as a standard part of the Transfer Control Protocol (TCP). There is also the danger of interfering with other household appliances, such as remote controlled garage doors or microwaves, as these also often use the 2.4GHz unlicensed frequency band. Using a wireless network eliminates this problem, as all network terminals are aware of others that might have an effect on them.

Conclusions

This project has achieved the goals it set out to reach, allowing multiple users to create, manage and access their own online music libraries through the combination of a specialised speaker unit and a separate web-based interface. The prototype speaker unit works correctly when in a suitable wireless environment, providing a simple yet versatile way for users to access and play music that they have previously uploaded to their library. It is also highly portable requiring nothing more than access to a suitable network and a power point in order to function.

The Setup Interface provides the functionality needed to allow the creation and development of users' libraries and thus make the Playback Interface useable. The underlying classes create and retrieve the meta-data that allows the system to provide users with accurate information on their libraries through both interfaces. This is most essential for the Playback Interface as the whole point of using Wireless Speakers is to be able to play music wherever and whenever you want it. It is therefore very important to see accurate lists of the different media that is available.

The most significant drawback of the system is the lack of file verification in the Setup Interface. Ideally, the system should only allow users to upload files that will actually be playable through a speaker unit as this eliminates the need for error checking at later stages and does not waste the user's time uploading files that can never be used. Being a web application, the system will never fail completely due to an error, however it is still possible to get an exception report returned by Tomcat instead of a web page returned by the servlet. Obviously it is undesirable for a user to be shown a Java Runtime Exception report instead of a more informative page reporting any problems that have occurred.

The Playback Interface would also have been more useful as a separate application that could be installed on Windows CE. This would allow the program to run in an offline mode if there was no network connection available and would also make future extension of the interface's functionality easier as it would not be dependent on outside technology such as the Windows Media Player Control for Pocket Internet Explorer. At present there is no way to implement features such as loop or shuffle play on an album, however this could be done if the Mobile Media API were fully implemented and the interface were to run as a Java application.

There are three main areas in which this project could be used and/or extended. Firstly, there is the use of a Wireless Speaker or multiple speakers as part of a home network. Anyone with a wireless network card in their home PC could use their PC as a server and then use the Wireless Speaker(s) as a portable stereo for their home and garden. This provides a direct comparison to the products discussed in the Evaluation section of this report (see above) as these products are all designed with home use in mind. A Wireless Speaker would still have advantages over these however, as it is more portable than the

multiple pieces of equipment needed for either the Sonic Link or Sound Power products, and the IRemote application is only a 'super-interface' does not support the remote playback of music files.

Although the multiple user support would not be essential in this case, it could still be useful as, for example, different members of a family could have their own units and thus their own default settings, playlists, and so on. It would also be very easy to adapt the system to allow all users access to all media files in this scenario, allowing a family to share their music as they would do with hard-copies (e.g. CD's) without the hassle of needing to obtain the physical medium that the music is on.

Secondly, it could be extended and made commercially available over the Internet. It is already in a state where it can be used globally over the Internet, however to be commercially viable it would need the addition of features such as a more secure login procedure to prevent unauthorised access to users' libraries. With the obvious issue of Internet file sharing to bear in mind, the system would also need to impose restrictions on access to its libraries to prevent people from collaborating to use it as a file sharing system. At the very minimum, this would entail allowing only one speaker unit access to a particular library at any given time, but to prevent the threat of legal action from record companies it might be necessary to only allow one speaker unit per user in total, which would also mean having a very reliable method of identifying speaker units. This would probably need to be some form of hardware recognition such as the serial number of the wireless card.

Finally, the application with the most commercial potential is to adapt the system to run on the new generation 3G mobile phones. This would make user identification very easy and also dispel any copyright issues over file sharing. Mobile phones are obviously more portable than a speaker cabinet and this system could become the online equivalent of a portable CD or Minidisc player. It would not necessarily be restricted to setting the phone up with other sound equipment to create up a kind of temporary stereo, but could also work with headphones to be a truly mobile music player.

This seems the most viable application for the project as mobile phone manufacturers are currently developing new technology faster than ideas of what to do with it. Portable music players are already massively popular and this technology would allow users to effectively carry their entire music collection with them, ready for instant access. A final extension to the system is the idea of replacing or complementing the file upload procedure with the facility to add music that the service provider already holds in storage and thus being able to 'buy' music from your service provider without having to upload it yourself.

This would, of course, require licensing agreements with record companies to allow distribution of their material in this manner, however some of these companies, such as EMI and the Sony Music Group are already pioneering ways to distribute their music directly over the Internet without requiring customers to buy a hard-copy of the music. It would seem that now is the perfect time to be exploring the market potential of such an application.

Appendices

Appendix A – Problem Definition & Objectives

Problem Definition

The aim of this project is to design and construct a speaker unit capable of playing music stored on a server without a physical connection to it, as well as software to run both the server and the speaker unit. It will use wireless ethernet and an iPAQ handheld PC embedded in a speaker cabinet to access, configure and play the sound files. As the system will work wherever the speaker unit is in range of an appropriate radio-ethernet transceiver, the unit is potentially globally portable, communicating with the server over the Internet.

The project will also include the creation of a databank containing the sound files themselves and meta-data for each file including track title, artist, album title, and data such as a track ID number that the system will use internally.

Objectives

Must Have's

- One example of a speaker unit that works in a wireless environment with the capabilities outlined in this document.
- Default playing of a pre-selected track/album following the iPAQ program being started up.
- Facility to remotely select a track/album from the server and play it through the unit.
- To create a user interface (for use on the server) with which the user can configure the server's default settings, can add or remove music files in the databank, and can give additional details as described in the problem definition.
- The server should be able to recognise different users according to which speaker unit is in use and provide the user with the choice of their personal settings etc., as well as any shared settings or files.

Should Have's

- Creation and modification of playlists through a user interface on the server, along with the ability to access and play these through the interface on the iPAQ.
- Facility to play an album/playlist through the unit that exceeds the memory capacity of the iPAQ.
- Basic extra functionality of an audio player, such as skip/search, random play, loop etc.

Could Have's

- Facility to play a single sound file through the unit that is larger than the available memory.
- To allow the user to create playlists remotely using a user interface on the iPAQ.
- To allow the user to create 'random' playlists by assigning each track or album a style and then requesting a random selection of tracks of a particular style or by a particular artist.

Appendix B – Project Specification and Plan

Wireless Speakers
Joint Honours Project

Julian Smith

Supervisor: Graham Kirby

Submitted: 30th October 2002

Contents

Introduction **2**
Context Survey **2**
Relevant Technology 2
Similar Available Products 3
Requirements Specification **6**
Functional Requirements 6
Non-Functional Requirements 9
Project Plan **9**
Modular Design 9
Implementation Strategy 12
Testing Plan 14
Fallback Plans 14
References 14
Project Monitoring Sheet **15**

INTRODUCTION

The aim of this project is to develop a system that allows the playing of music files, which are stored on a server, through a unit that is connected to wireless Ethernet, and is therefore potentially geographically independent from the server. One of the important points of this project is to produce an entirely portable and fully integrated unit. The products discussed in the context survey and illustrated in Fig.'s 1.1 - 1.3 below both require multiple pieces of equipment in order to first control, and then playback, music remotely from the server. A general requirement of this project, therefore, is to produce a single unit that interacts with the server both by controlling its music library, and by accessing the files themselves and playing them back to the user. An illustration of this is shown below:

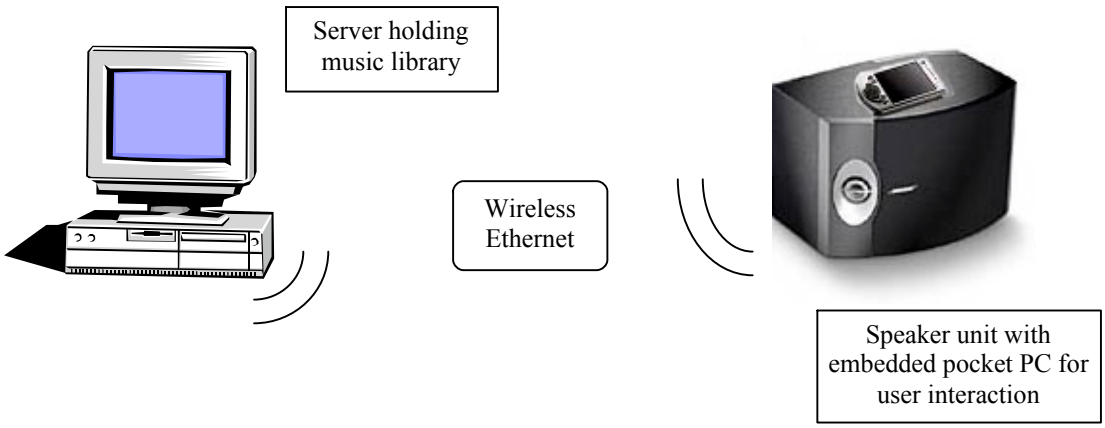


Fig 1.0 Illustration of the proposed system layout

CONTEXT SURVEY

Popularity of .mp3 Format [1]

The use of mp3 audio files is ever increasing. Portable mp3 players and mp3 playing software for desktop and laptop machines are becoming more and more common and diverse. Whilst there have been some controversial issues surrounding the use of mp3, for example napster.com, it remains perfectly legal to make copies of your music collection for personal use. Thus there are no barriers to an individual saving their CD's etc. onto their PC, and indeed adding to their collection with legitimately free music from the World Wide Web.

Pocket PC's

In the past few years handheld and pocket PC's have become widely available on the commercial market. Whilst their capabilities are limited compared with those of a desktop or laptop machine, the facility of having a general operating system on such a highly portable device has proved very popular.

Most pocket PC's are supplied with the Windows CE operating system, as well as stripped-down versions of some common Microsoft applications such as Word, Excel and, more importantly to this project, Internet Explorer and Windows Media Player. The user interface on the pocket PC will use these two applications for accessing and playing music files.

Java [2][3]

The use of Java in web technology has expanded rapidly since its introduction. The rise of object-oriented programming and Java's portability across platforms has contributed significantly to this. Particularly useful to this project are Java Servlets which provide good facilities for generating dynamic web pages, and Java Applets which can be used to create more interactive and powerful web pages than are possible with plain html.

Sun are currently developing the Mobile Media API (MMAPI)[4] for devices, such as pocket PC's, that use Java 2 Micro Edition development kit. I researched this as it provides facilities for audio playback which could have been used in this project. However only a reference implementation is currently available and no functionality will be implemented until June next year at the earliest. Details can be seen at: http://java.sun.com/products/mmapi/ .

Similar Available Products

As I stated in the introduction, there are many mp3-playing products available now on the commercial market. I have not found any that provide the same system that I am developing, however here are some that provide similar functionality from the user's point of view.

Turtle Beach Sonic Link Wireless MP3 [5][6]

Turtle Beach, a division of Voyetra inc., makes a product that allows the user to play mp3 or Windows Media Audio (.wma) files, through a home stereo. It does this with an 'Audio Sender' that attaches to the PC's line out, and an 'Audio Receiver' that plugs in to the home stereo. There is a remote control that interacts with the PC to control the playback of sound files. The figure 1.1 shows the layout of this system. The remote control has one-way communication with the PC, which in turn transmits to the stereo via the Sonic Link units.

The key difference between this product and my project is that the server transmits an analogue audio signal through the Audio Sender, which is then amplified whereas my project will be using file transfer or audio streaming, and the actual file decoding will be carried out on the pocket PC in the speaker unit. This will allow the user and the server to be completely geographically independent.

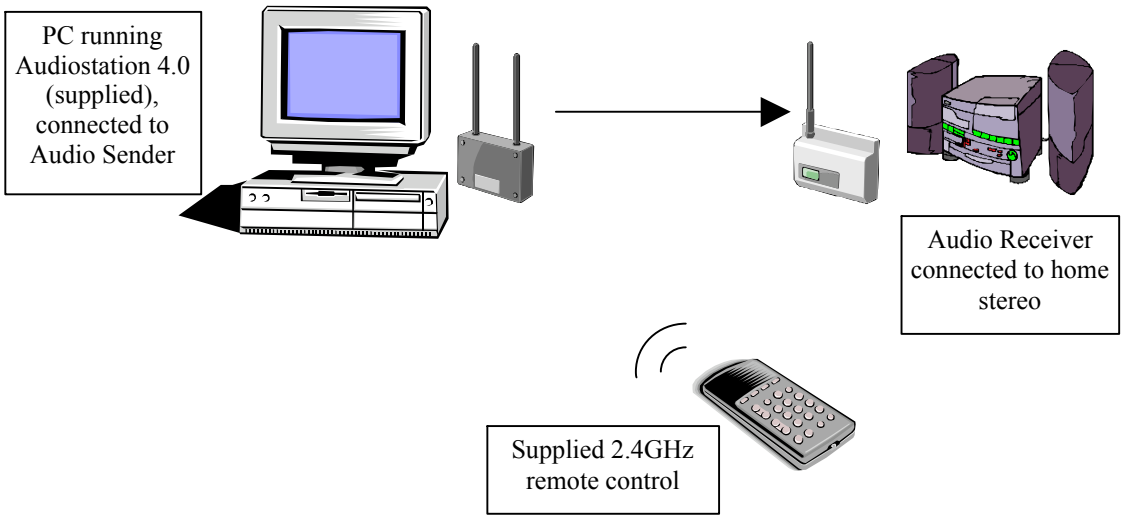

PC running Audiostation 4.0 (supplied), connected to Audio Sender
Audio Receiver connected to home stereo
Supplied 2.4GHz remote control

Fig. 1.1 Diagram representing the Sonic Link system

IRemote [7]

This is purely a software product that is more similar to my project than Turtle Beach's Sonic Link. It runs on Windows CE and is a 'super-interface' that allows the user to control the media player on their desktop or laptop machine by using a pocket PC as a remote control over wireless Ethernet. In itself it provides no method for playing music on a machine other than the server and extra equipment, such as a radio link, would be required to hear music being played in a different location. The range over which the pocket PC can be used is governed by the range of the wireless network.

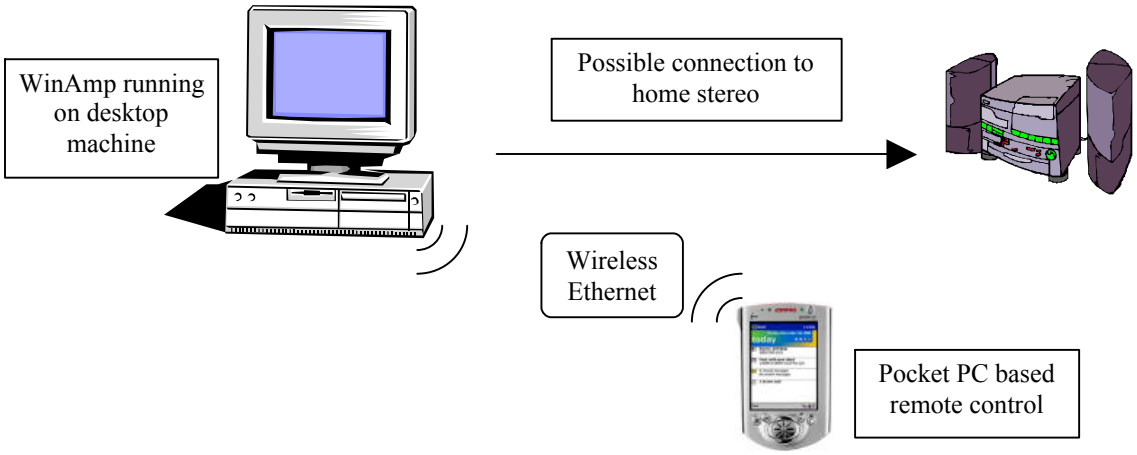

WinAmp running on desktop machine
Possible connection to home stereo
Wireless Ethernet
Pocket PC based remote control

Fig. 1.2 Diagram showing how IRemote can be used

The key difference between this and my project is that the pocket PC does not download files from the server and play them, instead it simply controls what the server's media player does, in much the same

way as the Sonic Link's remote control. The system that I am developing allows for the server to be completely remote so that the speaker unit can be taken anywhere in the world where there is a suitable wireless network available.

Wireless Sound Power Stereo Speaker Jack System [7]

Another similar product, available from 'www.x10.com', is the Wireless Sound Power system. This product consists of a 2.4GHz analogue radio transmitter and matching receiver with built-in amplifier, a remote control, and a PC receiver for the remote. Software that interprets the remote control's signal and can control media players such as RealJukeBox and WinAmp is freely available for download. The transmitter is connected to a PC's line out socket and the receiver to any hi-fi speaker or pair of speakers. The range of the music playback from the server is limited to 100 feet, which is the range of the radio transmitter.

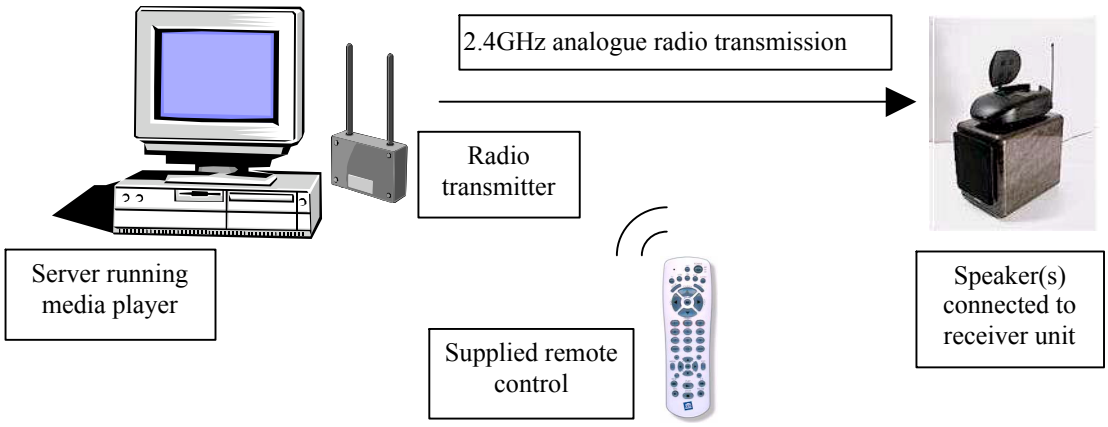


Fig 1.3 Diagram showing layout of Sound Power System

As with the two products above, this product relies on different equipment to control what is playing and to actually play back the music. The remote control allows the user to control a media player that is running on their PC. The radio transmitter then takes the analogue signal from the PC's line out socket and broadcasts it. The radio receiver picks up this signal, amplifies it and sends it out through the speaker(s). This means that the system is limited by the range of the radio transmission, both from the transmitter to the receiver and from the remote control to the PC receiver.

REQUIREMENTS SPECIFICATION

Definitions

The system - this refers to both the software and hardware components of the project.

Speaker unit - the physical unit that will play the music files. It includes all of the components listed in the Hardware Construction section of the Project Plan. From the users' point of view, it consists of a speaker with a touch screen that provides access to their personal music library.

Music library - the file space on the server where the users' music will be stored

FUNCTIONAL REQUIREMENTS

The functional requirements break down naturally into two components. These are firstly those available to the user by direct interaction with the server, and secondly the requirements for the speaker unit itself, including both software and hardware requirements.

Server

The server will store and manage the music library, support the server interface to allow for configuration of the system, and provide the user, via the speaker unit, with up-to-date information about the music library, as well as access to the files in it. The specific requirements for the server's functionality are listed below in the server interface section. Whilst the interface provides a method of access for the user, it is the server code that will actually implement the functionality outlined below.

Server Interface

This will be more complex than the pocket PC user interface to allow the user to configure the system to their own preferences. It will include screens allowing the user to carry out the following tasks, which are listed by priority. Each task will not necessarily have its own screen.

Essential Tasks

Adding Files to the Music Library

Users shall be able to save files into the music library, entering additional information as they choose to. This information includes track title, artist name, album title and style. This information is not required, but it must be entered for users to be able to search the music library using these attributes as parameters.

For convenience, the user shall be able to add an album on a single page in the interface. This will only allow one artist name and one style to be entered for all of the tracks on that album.

Default Choice of Music

Each user will have a default choice of music that can be played at the touch of a single button on the speaker unit. A user shall be able to change this default choice to any track, album or playlist in the music library. The server will not allow the user to remove the current default choice from the music library.

Expected Extensions

Creating Playlists

The user shall be able to create playlists of up to 20 tracks, in whatever combination or order that they like. This function can also be used for storing compilation albums in the music library as the feature for adding entire albums (above) will only support one artist and one style per album.

Modifying Playlists

Playlists can be modified by either removing individual tracks from the list, or adding them. Direct reordering of the list may be achieved by removing a track or tracks and then adding them in a different place in the list.

Deleting a Playlist

A playlist can be completely removed from the music library. This will not remove any of the tracks in the list from the library.

Possible Extensions

Availability of Files in the Music Library

Each user shall be able to select whether or not to make a track or album globally available (to all users) as they are saving it into the music library. Similarly, they shall be able to decide whether or not to make a playlist globally available as it is being created. Choosing this option requires that all of the tracks in it are also globally available. The system shall prompt the user to remove or change the permissions of any files that are restricted to their own personal use when creating such a playlist.

Adding Music Styles

The music library will hold a list of styles by which all tracks and albums can be categorised. Users shall be able to add styles to this list to provide for the tracks that they are entering into the music library. The style list will be globally available and thus will be the same for all users, whether or not there are tracks of every style available to each of them.

Speaker Unit

Pocket PC Interface

The user interface on the pocket PC should allow the user to carry out the following actions:

- Play a default choice of music (set by the user) at the touch of a single 'Play' button.
- Provide access to user-specific information such as their own selection of playlists or albums.
- Choose a track, album or playlist on the server from those currently available to the user and play it through the speaker unit.
- Create a playlist from the tracks currently available on the server and select whether to make this globally available or available only to themselves.
- Request a random playlist from the server, providing a style or artist and the number of tracks desired.

Essential Features

Playing of Default Music

When the interface is started, the user shall have the option of playing a default selection of music by pressing a simple 'Play' button, or entering the options pages as described below. The user can set this default selection through the server interface.

User-Specific Settings

Each speaker unit will act as a different user to the server. Thus each unit will have a different default selection of music, and potentially access to different playlists and music files. See the server interface section for details about user-specific playlists and music files.

Selection of Music from the Music Library

The interface will include pages that allow the user to select single tracks, albums or playlists that are available globally, or to them specifically. Only one of these can be selected at a time and the speaker unit will begin to play the selection automatically as soon as a choice is made.

Possible Extensions

Creation of Playlists

The interface will include the facility to create new playlists from the tracks available to the user. Modification and removal of playlists will only be possible through the server interface.

Requesting a Random Playlist

The user will be able to request a random playlist from the server, specifying the number of tracks that they would like and the style or artist that they would like to hear. If there are not enough tracks available to make the requested playlist then all those available will be played in a random order.

Hardware Construction

The hardware requirements for this project are minimal relative to the software requirements. The speaker unit will need to produce a reasonable sound quality, for example enough to fill the senior honours lab at a volume that is clearly audible. Consequently a small amplifier may need to be fitted to boost the audio signal from the pocket PC. This means that there must be sufficient room inside the speaker cabinet to accommodate the speaker cone, the amplifier, the pocket PC, including the expansion pack that houses the network card, and a suitable power adaptor for the pocket PC and the amplifier. For development purposes, the pocket PC will need to be easily removable from the unit. The unit will require access to a single power point in order to function and, other than its power cable, shall be completely self-contained.

NON-FUNCTIONAL REQUIREMENTS

Portability

The speaker unit shall be able to function wherever it is within range of a suitably configured radio-Ethernet (802.11) transceiver, and within reach of a standard 13amp power socket.

Usability

The user interfaces should be designed to allow for very easy operation of the system's basic functions, as well as having a clear structure that enables users to find other options quickly.

Reliability

The system shall not fail when playing the requested music at a rate of more than one in 20 operations.

Documentation of code

The server code will be documented with inline comments and javadoc documentation.

Documentation of User Manual

A full, online user manual shall be available through the server interface. A reduced version of this shall be available as part of the pocket PC interface. There will also be a printed version of the manual that shall not exceed 10 pages in length.

Acceptability

The acceptance test for this project shall include the following demonstrable features:

- A wireless unit that is capable of playing a user's default track or album from its server.
- Selection and playing of any track, album or playlist that is available to the user.
- Facility on server interface to set default choice of music.
- Facility to add files to the music library, supplying additional information as desired.

- Creation of a new playlist through the server interface.

Installability

The server software shall be portable to both Linux and Windows 2000/Me platforms. The pocket PC interface shall run only on Windows CE on a suitable handheld or pocket PC.

Serviceability

The system shall be designed so as to allow for future development, such as adding functionality to the server.

PROJECT PLAN

MODULAR DESIGN

As the system breaks naturally into five major components, these shall form the basic structure of the modular design. They are as follows:

- Music Library
- Server
- Server interface
- Pocket PC interface
- Hardware Construction

Music Library

The music library will hold a variety of data including the music files themselves as well as meta-data on each file and data about albums and playlists. The music files shall be stored in a tree system of directories. From a single directory representing the music library, there shall be a common directory where globally available music shall be stored, and then a directory that is specific to each user. Each of these directories shall then contain sub-directories for each artist whose material is stored in that part of the library. There shall also be a general directory that contains tracks whose artist information is unknown to the system.

Each album shall be stored in its own sub-directory. This directory will also include a text file containing the Track Identifiers for the tracks contained in the directory. In each of the user directories and the common directory there shall be a sub-directory containing information on the playlists that are currently available.

In the common directory and each of the users' directories, there shall be an XML file for each track stored in that part of the music library, which contains the following information:

- Track Identifier – an integer value from 00000 – 99999 that the system uses to uniquely identify each track. This value will not be available to the user as it is only used internally and altering it could corrupt the data in the music library.
- Track title
- Artist name
- Album title
- Track style

Playlists shall be stored as an XML file that contains the name of the playlist and a list of the track identifiers for the tracks in the playlist.

Server

The server software will be coded in Java and will be a web server. It will contain 2 main packages, dealing with server interaction and pocket PC interaction.

Server Interaction

This package of the server code will be responsible for managing the music library, providing dynamic web pages for the server interface, and handling user requests made through the server interface. It will use Java Servlets to dynamically create web pages based on the music files and other data that is currently available in the music library.

Pocket PC Interaction

The server shall create web pages that provide the user with accurate information about the music library. It will also handle the requests sent from the speaker unit. In general, this will mean sending a batch of files to the speaker unit, which Internet Explorer will in turn send to Windows Media Player to play through the unit.

Server Interface

The server interface will handle all of the configuration options for the user. It will be web-based, consisting of dynamically generated web pages, and form pages to allow updates to the music library. Each page in the interface will contain a menubar along the top of the page, which will be in a separate frame to ensure that users can easily navigate through the options screens. This menubar will contain buttons that link to the following pages:

- Main index
- Adding files to the library
- Selection of default music
- Playlist creator
- Online manual
- Quit

Pocket PC Interface

The user interface on the pocket PC will be constructed to utilise the version of Internet Explorer that is installed on the unit. It will consist of the following html web pages, including form pages for the submitting of information to the server:

- Start-up page – giving option to play default music or enter options
- Options index
- Music selection index
- Pages for selecting a track, album or playlist (one page each)
- Playlist creator
- Random playlist creator

Hardware Construction

The speaker unit will consist of a standard hi-fi speaker cabinet that is spacious enough to accommodate the speaker cone itself along with a small amplifier, the pocket PC including the expansion pack that houses the wireless network card, and a power module to provide power to the amplifier and the pocket PC.

Below is a flow-chart showing the structure of the unit:

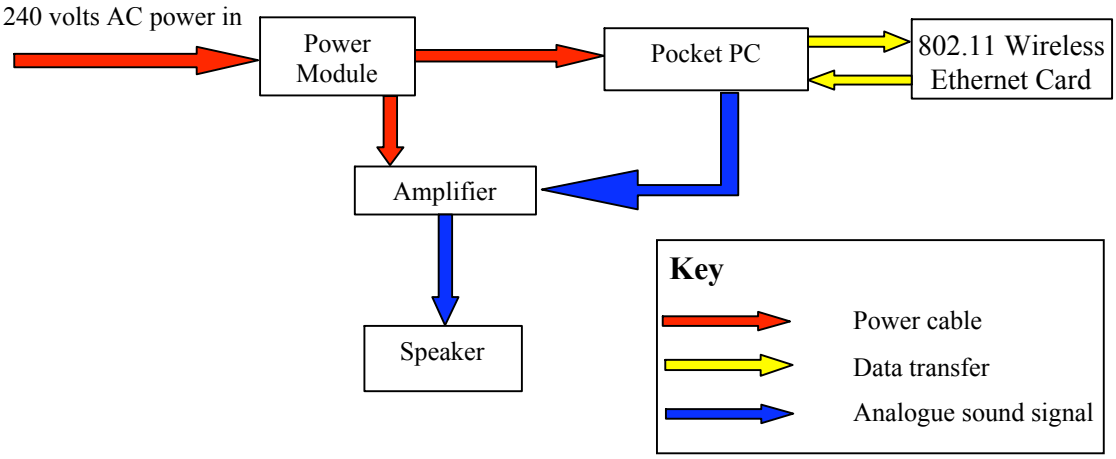


Fig 2.0 Diagram showing layout of the speaker unit

The power module and amplifier will be fitted to the inside of the base of the speaker cabinet. An aperture will be cut in the top of the cabinet and the pocket PC fitted underneath it. The opening will be the size of the pocket PC's screen, including access to the on/off switch.

IMPLEMENTATION STRATEGY

The pocket PC to be used for this project is a Compaq iPAQ H3660 running on Windows CE 2000 Edition. It has a basic version of Internet Explorer that will be used to display the user interface and Windows Media Player 8.0 that will be used to decode and play the music files transmitted from the server.

The software will be developed in a Windows environment, which is now available in the senior honours lab. This has been chosen because the synchronisation software for the iPAQ requires a Windows environment and it is far easier to write code on a desktop machine and upload it to the iPAQ than to write code directly onto the iPAQ. The server software will be designed to run on Windows 2000/Me and should also run on Linux.

All of the static web pages will be manually coded as html in a text editor to avoid the possibility of unwanted or unnecessary code being generated by a web page editor. This is particularly important for the iPAQ interface as the iPAQ has a very limited memory capacity and minimising file size is therefore crucial.

The Java code for the server will be developed in a Java development environment called JCreator[8]. This is available free on the Internet and provides all of the basic facilities for developing Java without the more complex functionality of an application such as Together.

TESTING PLAN

Server

The server code will be tested using a test harness that directly provides the text streams that the two interfaces will generate. The test harness will use examples that should both succeed and fail to test the following server functionality as it is implemented: adding files to the music library; setting a user's default choice of music; creating a playlist; modifying a playlist; deleting a playlist; adding a music style.

Interfaces

The server and iPAQ interfaces will be tested to ensure that all navigation links are correct and working. The form pages will also be tested with a test harness to make sure that they are producing the correct output for the server.

FALLBACK PLANS

The objectives for this project are classed in 3 categories: 'must have', 'should have' and 'could have'. It is expected that the first 2 of these sets should be implemented, barring any significant problems. The net result of a problem that puts the project irreconcilably behind schedule is therefore that only some

or none of the 'should have' objectives will be implemented. On the same token, if the project runs ahead of schedule then it is expected that some of the 'could have' objectives will be implemented.

There is an extra week left in the task schedule (see below) to allow for any unexpected delays in the project. If no delays occur then this week will be used to implement any additional features that are left from the 'could have' objectives.

REFERENCES

- [1] – The Destination for Digital Music
  - http://www.mp3.com/
- [2] – The Source for Java Technology
  - http://java.sun.com/
- [3] – Jia, Xiaoping – Object-Oriented Software Development Using Java
- [4] – Mobile Media API (MMAPI)
  - http://java.sun.com/products/mmapi/
- [5] - Turtle Beach SONICLINK Sonic Link, wireless MP3 -
  - http://shop.store.yahoo.com/digitally-unique/soniclink.html
- [6] – Article to PC world.com on Sonic Link product -
  - http://cssvc.compuserve.com/computing/cis/article/0,aid,37642,00.asp
- [7] – MCMajerees
  - http://www.mcmajeres.com/
- [8] - Wireless SOUNDPOWER Speaker Jack System -
  - http://www.x10.com/products/x10_vk59a.htm
- [9] – JCreator – Java IDE
  - http://www.jcreator.com/

PROJECT MONITORING SHEET

- Tasks Completed To Date

Problem definition & project objectives: Submitted Friday 11th October '02

Project Specification & Plan: Submitted Wednesday 30th October '02

- Task Scheduling for Remainder of Project

Task	Start Date	Finish Date
Server – low level design and prototype implementation	04/11/02	06/12/02
Testing with test harness and basic server interface	07/12/02	20/12/02
iPAQ interface basic design	07/12/02	20/12/02
Server – implementing additional features	03/02/03	16/02/03
Server interface – added features	17/02/03	23/02/03
iPAQ interface – added features	24/02/03	02/03/03
Interface testing	03/03/03	09/03/03
Hardware acquisition & construction	10/03/03	14/03/03
System testing	17/03/03	28/03/03
Writing of user manual	31/03/03	07/04/03
Collating and finishing final report	07/04/03	16/04/03

Deadlines

Submission of Requirements Specification and Project Plan	30/10/02
Interim Report 1	04/12/02
Interim Report 2	12/03/03
Project Report, Software and Documentation	23/04/03
Presentation	12/05/03

Appendix C – Interim Reports

Interim Report 1

Progress

The project is on schedule to date. Tomcat, a web server program that is capable of supporting Java Servlets, has been set up and tested. This is installed on Lochside in the Senior Honours Lab. Layout design for the iPAQ interface has begun, including how this will be implemented using Servlets and either form pages or Applets. I am currently researching which of these will interact more easily with Servlets for the purposes of the project. A Java plug-in for Pocket Internet Explorer has been installed and tested on the iPAQ to allow for the possibility of using Applets in the interface.

Problems

Since discovering that the Java MMAPI will not be useable until after the project has finished, there has been the problem of trying to keep the user interface on the iPAQ within one program as the initial fix for this included using Internet Explorer and Windows Media Player separately. There has also been a problem with the iPAQ's procedure for downloading Internet media content including audio files. Pocket Internet Explorer cannot be configured to automatically download and play media content by following links to media files on web pages. Instead it provides a prompt box asking if the user would like to save the file to main memory. There is an option to play the file after download but this would be a very clumsy system to have to use every time a user wanted to access some music from their library.

I have been unable to test any basic Servlets as I do not yet have the Servlet library package and the current Internet blackout affecting the department has prevented me from downloading it from the Sun Java website. This can be done once Internet access has been restored or the package is obtained from elsewhere.

Solutions

The solution to both of the problems above has been found in the use of the Windows Media Player Control for Pocket Internet Explorer. This is a piece of software that allows Windows Media Player to be embedded into pages displayed by Pocket Internet Explorer. In addition, Microsoft JScript can be used to remove the standard Media Player interface from view and create customised buttons, meaning that the page need only contain the exact Media Player functionality that is required by the user. This control software also allows for the preloading of media files that are specified by the page. Servlets can therefore be used to dynamically create web pages that specify the appropriate media files from the library and include JScript instructions to provide only the essential Media Player functionality that the page needs.

Interim Report 2

Progress

The software to generate dynamic web pages for the iPAQ is now mostly complete. The system is able to recognise different users using cookies and offer a signup facility to new users. The home page for each user can be loaded using their default choice of music although some problems still exist with the transfer of audio files (see below for details). Users are able to choose tracks, albums and playlists from their music library as per the project brief. The interface for desktop machines is at a more basic stage still, but it should have the functionality to upload tracks & albums, including meta data for these, and to change the default music within a week.

This interface is where most work still needs doing. The most significant part of this is the underlying code to handle uploading files to the server via the Internet. Once this is complete, all of the essential requirements for the project(with the exception of a prototype unit) will have been met as well as some non-essential ones. I have moved the construction of the prototype unit back in the schedule to allow more time now for further software development.

Problems

There are currently problems with the transfer of audio files from the server to the iPAQ. The Media Player Control reports that it cannot play a file because it could not be found or is not valid. This is a confusing problem as it has worked on occasion and then spontaneously stopped working without any alteration to the server code. I hope to get an answer from Microsoft soon that might help explain why this is. It seems to be purely a network problem as the control software works perfectly with files that are stored in memory on the iPAQ.

Solutions

As a backup to the problem with audio file transfer, I will be testing some Microsoft JScript that works in conjunction with the control software to see if this manages to download audio files successfully. The worst-case scenario from this will be to go back to using Windows Media Player separately from Internet Explorer by dynamically creating links to audio files. However this would be significantly more awkward for the user as it involves switching between two programs on the iPAQ and so I hope to avoid it.

Appendix D – Testing Summary

Programming Procedures

Testing a web application is fundamentally different from testing a normal application, as the standard output of a servlet class is a dynamically generated web page. Therefore, the only way to properly check the servlet output is by viewing the page source of the pages returned. It was also unsuitable to use a test harness of any sort for this as the classes need to be tested in a servlet context, i.e. running on a web server and it is simplest to just browse the servlet's pages with every possible combination of parameters.

The first part of the system to be built was the basic function of the Playback Interface – getting the pocket PC to download and play some music from the server. This is the most crucial part of the project as, if it does not work properly, the rest of the system is useless. The basic testing principle during development was the reverse of what often happens. It involved developing and testing the code that would allow the user to make use of their library and then working backwards to the point of constructing the library in the first place. Testing during development was therefore based on testing every option of an action, then adding the previous action and testing this right through every possibility. For example, the basic mechanism for playing music through the iPAQ was written and tested, and then the facility of selecting a track from a dynamic list was added and both procedures tested in sequence.

This process was followed until a complete system was developed from uploading some music, to sorting and displaying it, and ultimately to playing it through the Playback Interface. The system was then expanded horizontally, adding the code for dealing with albums and then playlists, with the same testing procedure applying to all steps. The extra features such as changing the default music were then added and tested, firstly on their own, and then with the other features in the system.

System Testing

Once the system was complete and in its final form, the server was left online and beta testing was requested from other members of the class. This testing showed up some problems with the effectiveness some of the JavaScript across different platforms and web browsers. As a result of this, some extra parameter checking was included in several of the Setup Interface pages, and also the default selecting of list options to prevent the submission of an unexpected null value when the JavaScript failed.

Further testing has highlighted the lack of file verification in the Setup Interface, however this seems to be the only significant drawback to the system. In particular the Playback Interface is very stable as most of its functionality is based on what the server knows is available and there are very few places where the user has the opportunity enter potentially corrupting input.

Appendix E – Status Report

Table Showing Original Objectives and Their Level of Completion

Objective	Status
Example of a speaker unit that works in a wireless environment	Achieved
Default playing of a pre-selected track/album on start-up of the Playback Interface	Achieved
Facility to remotely select a track/album from the server and play it through the speaker unit	Achieved
Creation of a user interface (Setup Interface) that can be used to change default settings, add/remove music files and meta-data.	Achieved
Recognition of different users according to which speaker unit is in use.	Achieved
Creation and modification of playlists through the Setup Interface, and access to them through the speaker unit.	Achieved, except modification of playlists is only possible through re-creating/removing and replacing them.
Facility to play an album/playlist that exceeds the memory capacity of the iPAQ.	Achieved
Basic extra functionality of an audio player such as skip/search, random play, loop, etc.	Achieved, except search not possible as media files are streamed, loop not possible. These are limitations of the technology in use.
Facility to play a single file that exceeds the memory of the iPAQ.	Not tested as a suitable file not yet found.
Creation of playlists through the Playback Interface.	Achieved
Creation of random playlists, by style/artist and size.	Achieved

The table above shows the final status of each of the original project objectives at the end of development. Even though the mechanism for accessing music files has changed since the original project definition, these objectives have not needed to change to compensate for this and the level of their achievement demonstrates that the alternatives method employed has worked equally well from a usability perspective.

Appendix F – Maintenance Document

Setting Up Wireless Speakers

The files needed for Wireless Speakers are split into two sections. Firstly there are the Java classes, which need to be compiled into the correct location in the web server's file structure. In the case of Tomcat, this path is 'ROOT/WEB-INF/classes/', however in their servlet context, the servlet classes are accessed at 'ROOT/servlet/ServletName'. In addition to the Java files, there are some static files which should be placed directly into the root directory. These include the graphics files for the Playback Interface, the online help files for both interfaces, and some other system files such as 'users.txt'.

Some of the constants in the Values class will need to be changed if a different server is in use. The server name will of course be different, and the relative path between the servlet classes and the web root directory may be different depending on the server being used. The application can only be tested in a servlet context so testing will involve carrying out all the possible actions provided by the servlet classes and Tomcat will return an exception report if there are any run-time errors, which can be analysed to track down the bugs that caused them.

Wireless Speakers will be permanently online at http://inverleven.dcs.st-and.ac.uk until after the project demonstrations in mid-May. There cannot be a specific testing strategy for this kind of application so the best way to test it is simply to use it and try out all of its features.

It should be very easy to add functionality to either interface by creating new servlets that provide the extra features. Of course, any new code will need to be tested for consistency to ensure that it does not create the potential to corrupt the media files or meta-data in a user's library.

Return-Path: ron@dcs.st-and.ac.uk
Delivery-Date: Wed May 21 14:02:30 2003
Return-Path: <ron@dcs.st-and.ac.uk>
Delivered-To: sal@localhost.dcs.st-and.ac.uk
Received: from localhost (localhost.localdomain [127.0.0.1])
        by caolila.dcs.st-and.ac.uk (Postfix) with ESMTP id C0CE236FE3
        for <sal@localhost>; Wed, 21 May 2003 14:02:30 +0100 (BST)
Received: from chrystal.mcs.st-and.ac.uk [138.251.192.246]
        by localhost with POP3 (fetchmail-6.1.0)
        for sal@localhost (single-drop); Wed, 21 May 2003 14:02:30 +0100
(BST)
Received: from pittyvaich.dcs.st-and.ac.uk (pittyvaich.dcs.st-and.ac.uk
[138.251.206.55])
        by mcs.st-and.ac.uk (8.12.8/8.12.8) with ESMTP id h4LD7JY9013317
        for <sal@chrystal.dcs.st-and.ac.uk>; Wed, 21 May 2003 14:07:19
+0100
Received: from [138.251.206.113] (rhum [138.251.206.113])
        by pittyvaich.dcs.st-and.ac.uk (8.12.6/8.12.6) with ESMTP id
h4LD21FY021838;
        Wed, 21 May 2003 14:02:01 +0100 (BST)
Mime-Version: 1.0
X-Sender: ron@pop.dcs.st-and.ac.uk
Message-Id: <a05101002baf126240353@[138.251.206.113]>
Date: Wed, 21 May 2003 14:02:51 +0100
To: sal@dcs.st-and.ac.uk
From: Ron Morrison <ron@dcs.st-and.ac.uk>
Subject: Julian Smith
Content-Type: text/plain; charset="us-ascii" ; format="flowed"
X-ASK-Info: Whitelist match

Understanding of the Problem A
        Good discussion
Proper Software Engineering Process (including Plan) A
        Carried out the work well
Achievement of main objectives A
        He achieved all the objectives
Structure and Completeness of the Report A
        It is well presented
Structure and Completeness of Presentation A
        This was very good

ADDITIONAL CRITERIA
Knowledge of the literature B
        He showed a good understanding of the area
Critical evaluation of previous work B
        Limited but what he had done was good
Critical evaluation of own work
Justification of design decisions A
        A weird project well defended
Solution of any conceptual difficulties A

```
        The project worked in full
Achievement in full of all objectives                                   A
        It worked
Quality of Software                                                     A
        Looks without bugs
Ambition and Scope of Project                                           B
        Strange idea

EXCEPTIONAL CRITERIA
Originality of concept, design or analysis                              B
        There are variants
Adventure                                                               B
        Adventurous but not difficult
Inclusion of publishable material                                       B
        Maybe

Overall Grade   17

--
```


```
================================================================================
Ron Morrison, School of Computer Science,
University of St Andrews, North Haugh, St Andrews, Fife KY16 9SS, Scotland
Phone: +44 1334 463254, Fax: +44 1334 463278
e-mail:  ron@dcs.st-and.ac.uk WWW
http://www-ppg.dcs.st-andrews.ac.uk/People/Ron/
================================================================================
```

Replied: Tue, 20 May 2003 13:15:56 +0100
Replied: Graham Kirby <graham@dcs.st-and.ac.uk>
Return-Path: graham@dcs.st-and.ac.uk
Delivery-Date: Thu May 15 17:00:24 2003
Return-Path: <graham@dcs.st-and.ac.uk>
Delivered-To: sal@localhost.dcs.st-and.ac.uk
Received: from localhost (localhost.localdomain [127.0.0.1])
        by caolila.dcs.st-and.ac.uk (Postfix) with ESMTP id 3CE21379B4
        for <sal@localhost>; Thu, 15 May 2003 22:00:22 +0100 (BST)
Received: from chrystal.mcs.st-and.ac.uk [138.251.192.246]
        by localhost with POP3 (fetchmail-6.1.0)
        for sal@localhost (single-drop); Thu, 15 May 2003 23:00:22 +0200
(CEST)
Received: from pittyvaich.dcs.st-and.ac.uk (pittyvaich.dcs.st-and.ac.uk
[138.251.206.55])
        by mcs.st-and.ac.uk (8.12.8/8.12.8) with ESMTP id h4EEQAY9006167
        for <sal@chrystal.dcs.st-and.ac.uk>; Wed, 14 May 2003 15:26:10
+0100
Received: from [138.251.206.202] (edradour [138.251.206.202])
        by pittyvaich.dcs.st-and.ac.uk (8.12.6/8.12.6) with ESMTP id
h4EEHrFY019088;
        Wed, 14 May 2003 15:17:53 +0100 (BST)
Mime-Version: 1.0
X-Sender: graham@pop.dcs.st-and.ac.uk
Message-Id: <p06001204bae7bd1487e0@[138.251.206.202]>
In-Reply-To: <20030425135556.1D1C337970@caolila.dcs.st-and.ac.uk>
References: <20030425135556.1D1C337970@caolila.dcs.st-and.ac.uk>
Date: Wed, 14 May 2003 15:18:05 +0100
To: Joy Thomson <joy@dcs.st-and.ac.uk>
From: Graham Kirby <graham@dcs.st-and.ac.uk>
Subject: SH Project assessment - Smith
Cc: sal@dcs.st-and.ac.uk
Content-Type: text/plain; charset="us-ascii" ; format="flowed"
X-ASK-Info: Whitelist match

Spervisor's assessment of Julian Smith's Joint Honours SH Project
(CS4098):

Assessment

This is an excellent Joint Honours project which achieved all its
main objectives. The final product, report, code and presentation are
all of high quality.

Comments on product:

The finished product has been demonstrated to work well, with a
simple and effective user interface. The main limitations - not

working with MP3 files, and lack of verification of uploaded files - would be significant in a commercial product but do not really matter here.

Good online help is provided.

Comments on report:

This is well written and clearly presented, although it might have been easier to read if the speaker unit had been described before the server structure.

Comments on code:

Well formatted, good use of comments and JavaDoc documentation.

Comments on presentation:

Good slides, spoke confidently and held the attention of the audience. Use of video feed to illustrate speaker user interface worked well.

BASIC CRITERIA

Understanding of the Problem	Excellent
Proper Software Engineering Process (including Plan)	Good
Achievement of main objectives	Excellent
Structure and Completeness of the Report	Excellent
Structure and Completeness of Presentation	Excellent

ADDITIONAL CRITERIA

Knowledge of the literature	Good
Critical evaluation of previous work	Excellent
Critical evaluation of own work	Excellent
Justification of design decisions	Good
Solution of any conceptual difficulties	Good
Achievement in full of all objectives	Excellent
Quality of Software	Excellent
Ambition and Scope of Project	Excellent

EXCEPTIONAL CRITERIA

Originality of concept, design or analysis	Good
Adventure	Excellent
Inclusion of publishable material	Nothing obvious

--

Graham Kirby
School of Computer Science
University of St Andrews
North Haugh
St Andrews phone: +44 1334 463240
Fife KY16 9SS fax: +44 1334 463278
Scotland http://www-systems.dcs.st-and.ac.uk/~graham/